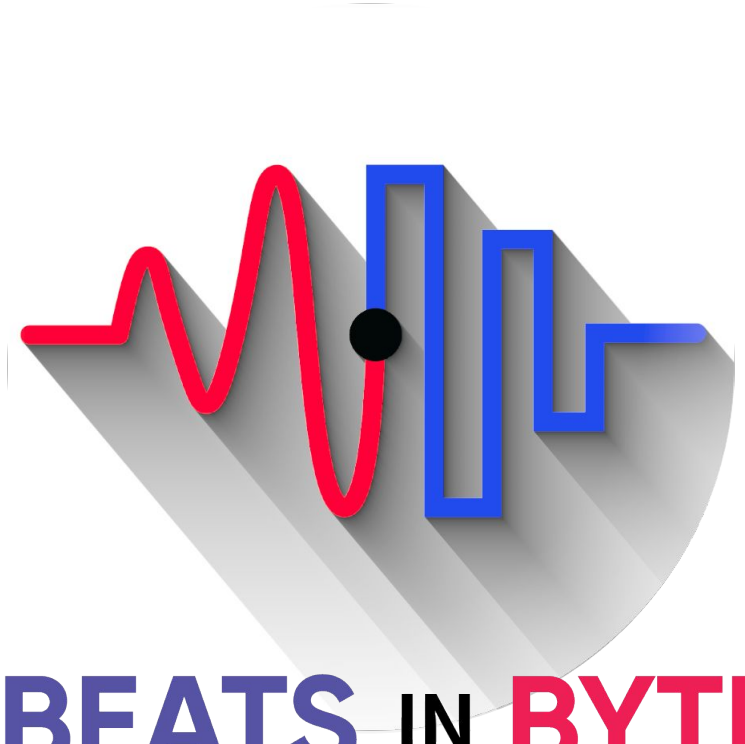


BEATS IN BYTES

Table Of Contents

Table Of Contents (Cont.)

Section 2: Data Analysis

Table Of Contents (Cont.)

Introduction

The goal of Beats in Bytes is to quantify and numerically interpret classical music. This begins by taking specific measurements from each song. We accomplished this by writing programs for data collection. The next step was to analyze our data so that we can understand exactly what statistical patterns exist in classical music.

Background Research

As we were dealing with them frequently during our data collection process, we needed to know more about music files. Music files are different from audio files - the former contain data that is organized like classical music, whereas the latter is just a collection of wavelengths.

We predicted we would find distribution laws in our data. One such law would be Zipf's Law (see *Zipf's Law*). Another distribution law is the Pareto Principle. The most famous example of this law is the 80/20 rule. It states that 20% of the causes are responsible for 80% of the effects. An example of this in our data would be that 80% of the songs in a certain time signature (an aspect of a song which each song has only one of) would come from a single time period (20% of our data). Another famous distribution law is Benford's Law. It states that for usage rank n, then the percent of the data in that value can be defined as log(n)-log(n-1).

Background Research (Cont.)

For our statistical analysis, we had to learn new statistical concepts. One of these was standard deviation, which is a measure of variation in data found by squaring distances from the mean. The formula to find standard deviation is:

$$s = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N-1}}$$

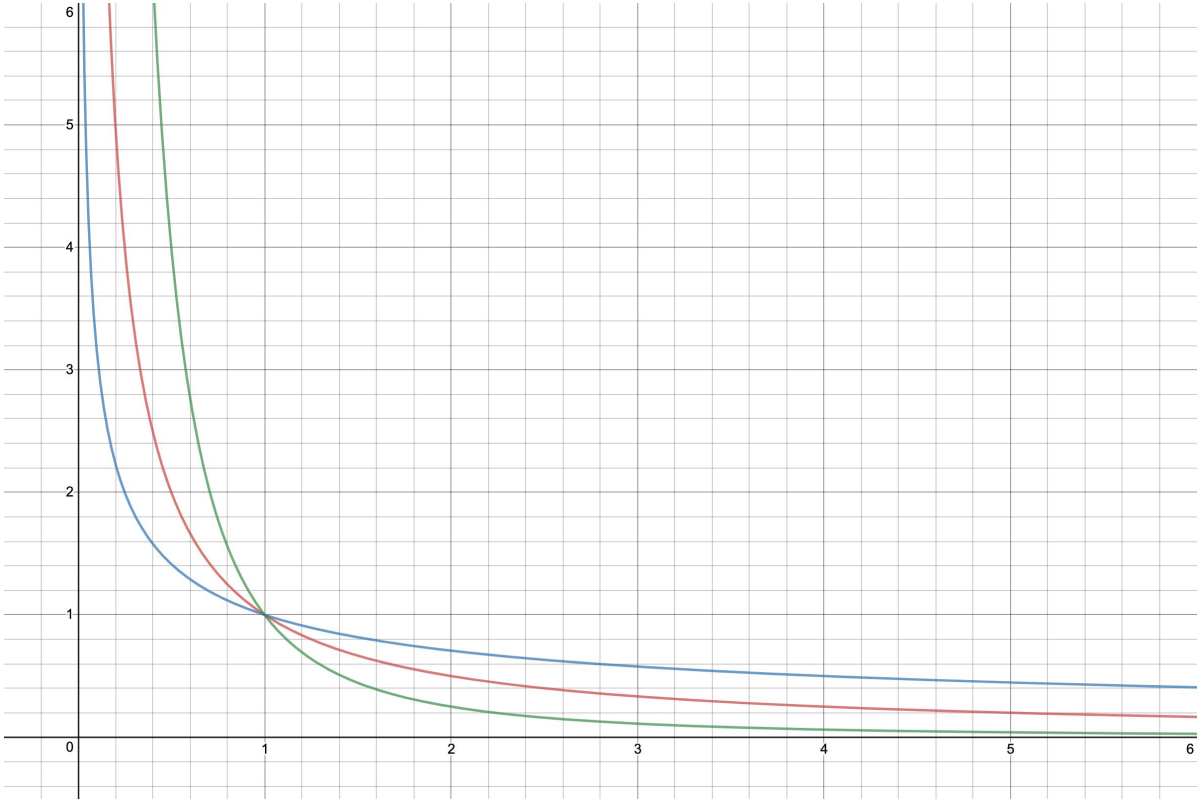
We also learned about t-Tests. A t-Test is a method of statistical comparison between two data sets that outputs a *p* value. If the p-value is smaller than 0.05, the null hypothesis (that the two data sets are from the same larger population) can be accepted. Also important to our project was the concept of correlation coefficients, or *r* values. A correlation coefficient is a measure of how well an equation can be used to represent a relationship between a set of x and y values. The formula for finding the *r* value for a linear regression (nonlinear is much more complex) is:

$$r = \frac{1}{n-1}\sum\left(\frac{x_i - \bar{x}}{s_x}\right)\left(\frac{y_i - \bar{y}}{s_y}\right)$$

Zipf's Law

<u>This is simply an explanation of Zipf's Law, and has nothing to do with our actual results or analysis.</u>

Zipf's law is a power curve that consistently shows up in many areas of life, most notably literature, where the usage of words has been shown to follow Zipf's Law quite well.
The equation that represents Zipf's Law is: $y = \frac{n}{x^a}$ where *x* is the usage rank, *n* is the largest value of the dataset, and *a* can be anything, but is 1 in a perfect Zipf's Law.



Red: $y = \frac{1}{x^1}$

Blue: $y = \frac{1}{x^{\frac{1}{2}}}$

Green: $y = \frac{1}{x^2}$

When fitting our data to Zipf's Law, we found the optimal *a* value for the regression, then compared the curves.

Section 1:

Data Collection

Meaning of “Metric”

Throughout the documentations of this project’s findings, the term “metric” is used quite frequently. In general, a metric is defined as “a system or standard of measurement.” Our usage of this term still follows this definition, but puts it into musical context. The meaning of “metric” for our purposes is better defined as an aspect of music. For example, average note value is a metric.

All Analyzed Metrics

I. Key Signature
II. Time Signature
III. Average Note Value
IV. Percent of Song In (including dotted versions of notes)
    A. Whole notes
    B. Half notes
    C. Quarter notes
    D. Eighth notes
    E. Sixteenth notes
    F. Thirty-second notes
    G. Sixty-fourth notes
V. Sequences of note with the same note value
    A. Number of such sequences 3, 4, 5, 6, and 7 or more notes long
    B. Length of longest of such sequences
    C. Most repeated note value
VI. Rhythmic Themes
    A. Total number of themes 2, 3, 4… 45 notes long
    B. Largest number of repetitions of themes 2, 3, 4… 45 notes long
    C. Total number of repetitions of themes 2, 3, 4… 45 notes long
VII. Most Used Note Values (measured in both occurrences and time in beats)
    A. 1st, 2nd, 3rd… 7th most used note values
    B. Percent of song in 1st, 2nd, 3rd… 7th most used note values
VIII. Most Used Pitches
    A. 1st, 2nd, 3rd… 7th most used pitches
    B. Percent of song in 1st, 2nd, 3rd… 7th most used pitches
IX. Average Pitch (relative to tonic)
X. Sequences of notes with the same pitch
    A. Number of such sequences 3, 4, 5, 6, and 7 or more notes long
    B. Length of longest of such sequences
XI. Average Steps per Jump
    A. Absolute value
    B. Including negatives
XII. Number of steps between first and last note
XIII. Number of scales of 2, 3, 4… 7 or more notes long
    A. Ascending
        1. Whole step
        2. Half step
    B. Descending
        1. Whole step
        2. Half step

Part A:

Music Files

Kern

```
**kern
*k[f#c#g#]
*M3/4
=1
4r
.
32c#/LL
323/
32a/JJ
32cc#\LLL
32ee\
32aa\
32ccc#\JJJ
4eee'\
.
4eee'\
=2
(8eee\L
16gg#\Jk)
16r
5gg#/
.
4aa
.
```

Fig. 1a: Kern encoding

(Left) A sample of ‘kern’, a way to represent music as text. It uses ‘spines’, which represent different parts of music that play simultaneously. The left is an example of a single-spine song, meaning it has only 1 instrument playing at a time. We defined this as a melody.

(Below) The standard musical notation for the displayed kern text on the left.

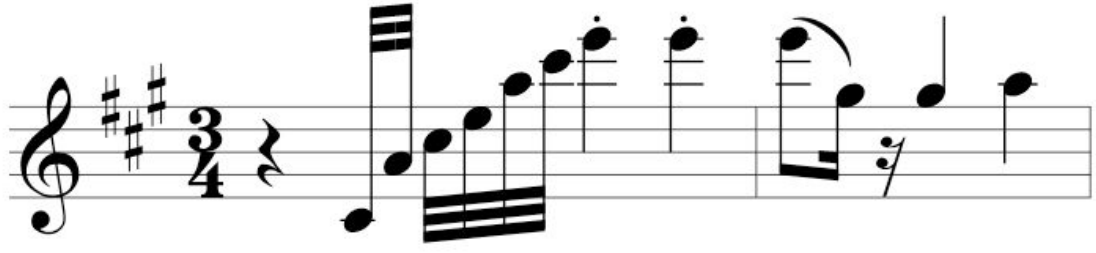
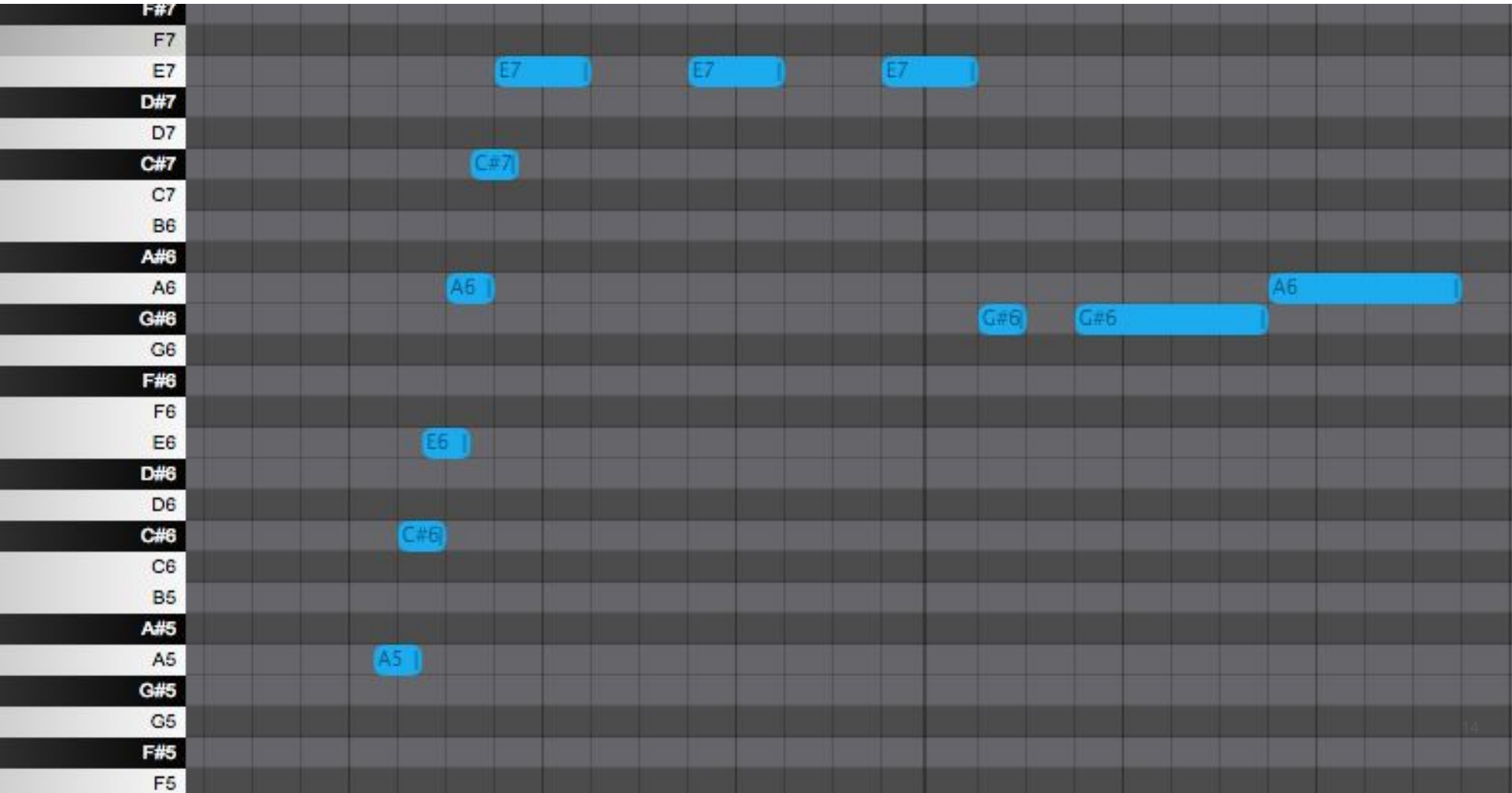


Fig. 1b: Musical Notation

MIDI

Below is a sample of the MIDI format, in *piano roll* notation. This snippet of music represents the same song as the earlier two examples. MIDI encodes each note with a timestamp, allowing easy playback. MIDI is the most common music file format, and allows for good visual representation. Since it is a common file format, we found our files as MIDIs, then converted them to kern (text) files.



Kunstderfuge (Website)

Kuntsderfuge, which means "Art of the Fugue" in German, was the name of the enormous MIDI file library from which we collected files. Below is a small sample of their massive collection. Each of these links downloads a song in the MIDI file format.

Gabrieli, Giovanni

Venezia c. 1555 - Venezia 1612

Choral music:

» MIDI | Hodie Christus natus est for 2 choirs SSATB + ATBBB
» MIDI | Jubilate Deo for choir SSAATTBB
Free sheet music collection of the W. Icking Archive, © Ulrich Alpers

» MIDI | Canzon à 12 for 3 instrumental choirs SATB+SATB+SATB
» MIDI | Canzon XII (1615) for 2 instrumental choirs SATB+SATB
» MIDI | Canzon [secunda] for 2 instrumental choirs SAAB+SAAB
» MIDI | Sonata Pian e Forte for 2 instrumental choirs SATB+TTTB
» MIDI | Canzon seconda a quattro (1608) *
Free sheet music collection of the W. Icking Archive, © Ulrich Alpers
* Arranged for board instruments

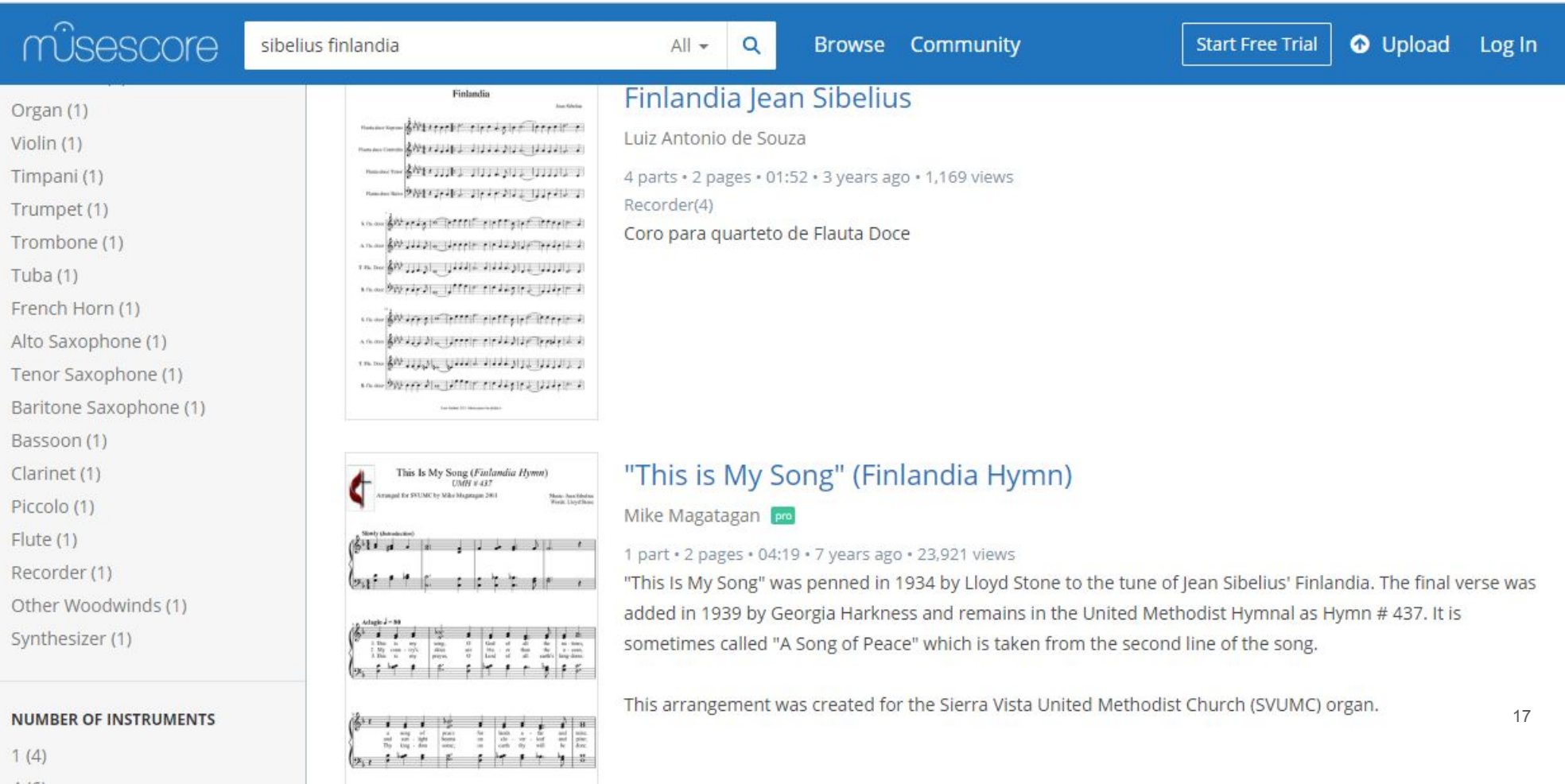
» MIDI | Domine exaudi orationem meam for 2 choirs SSAT+TBBB
» MIDI | Jam non dicam vos servos for 2 choirs SSAB+SSAB
» MIDI | O magnum mysterium for 2 choirs SSAT+ATBB
» MIDI | Plaudite omnis terra for 3 choirs SSAT+SATB+ATBB
» MIDI | Jubilate Deo, omnis terra for 11 instruments/voices
Free sheet music collection of the W. Icking Archive, © Fritz Brodersen

KernScores (Website)

KernScores was where we first started finding songs. While the website itself didn't have many songs, the songs it did have were already in kern, eliminating the file conversion process. Of the three websites we used for file collection, this one was the one that we used the least.

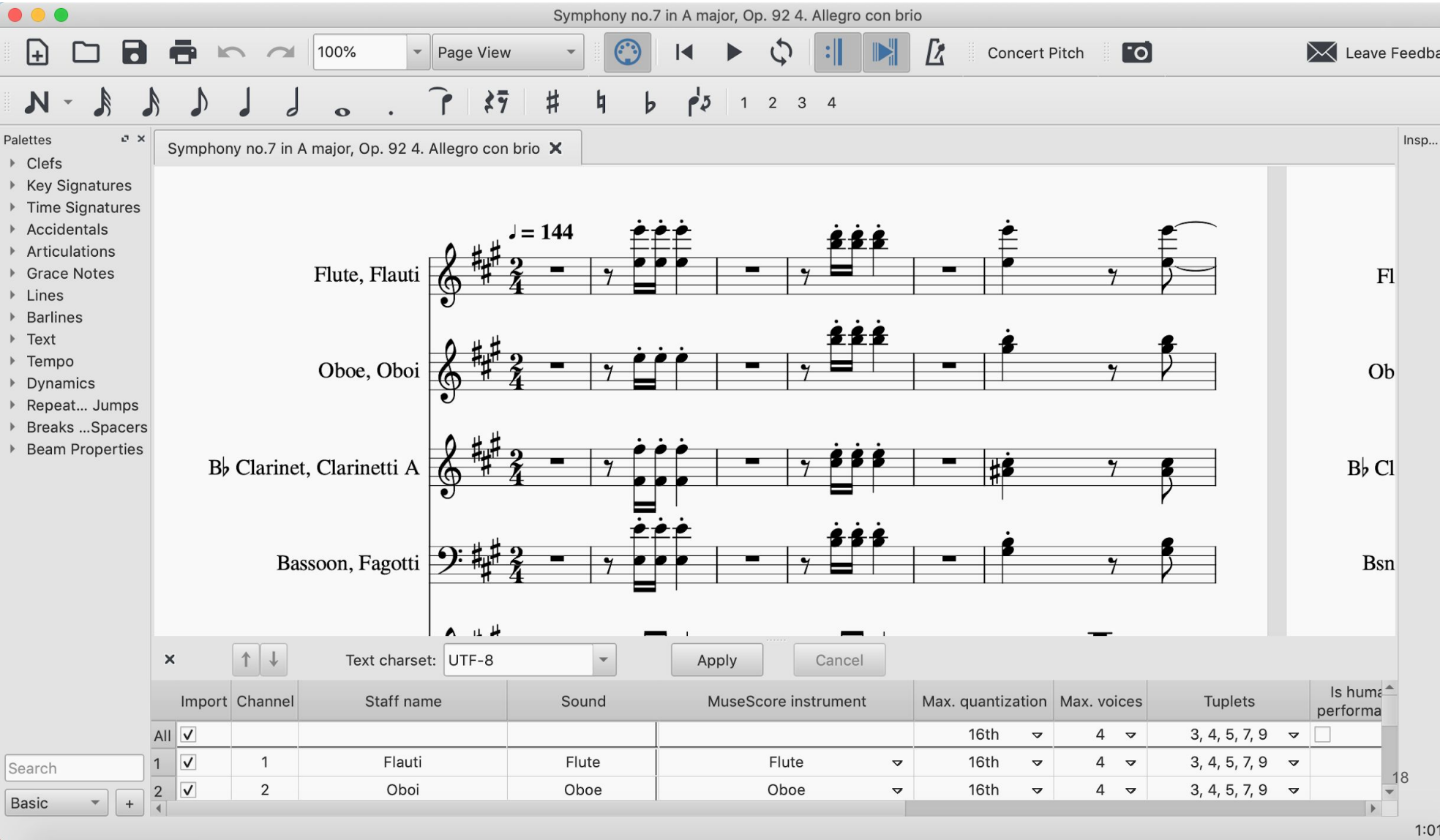
Musescore (Website)

The website *musescore* is a website where you can share sheet music and songs in various file formats, including MIDI. When kunsterfuge lacked enough songs for our composers, we looked here to download files.



Musescore (Application)

The music editing application *Musescore* (related to the *musescore* website) was used for converting MIDI files to musicXML. Our (somewhat) convoluted process then allowed us to convert this to kern (text).

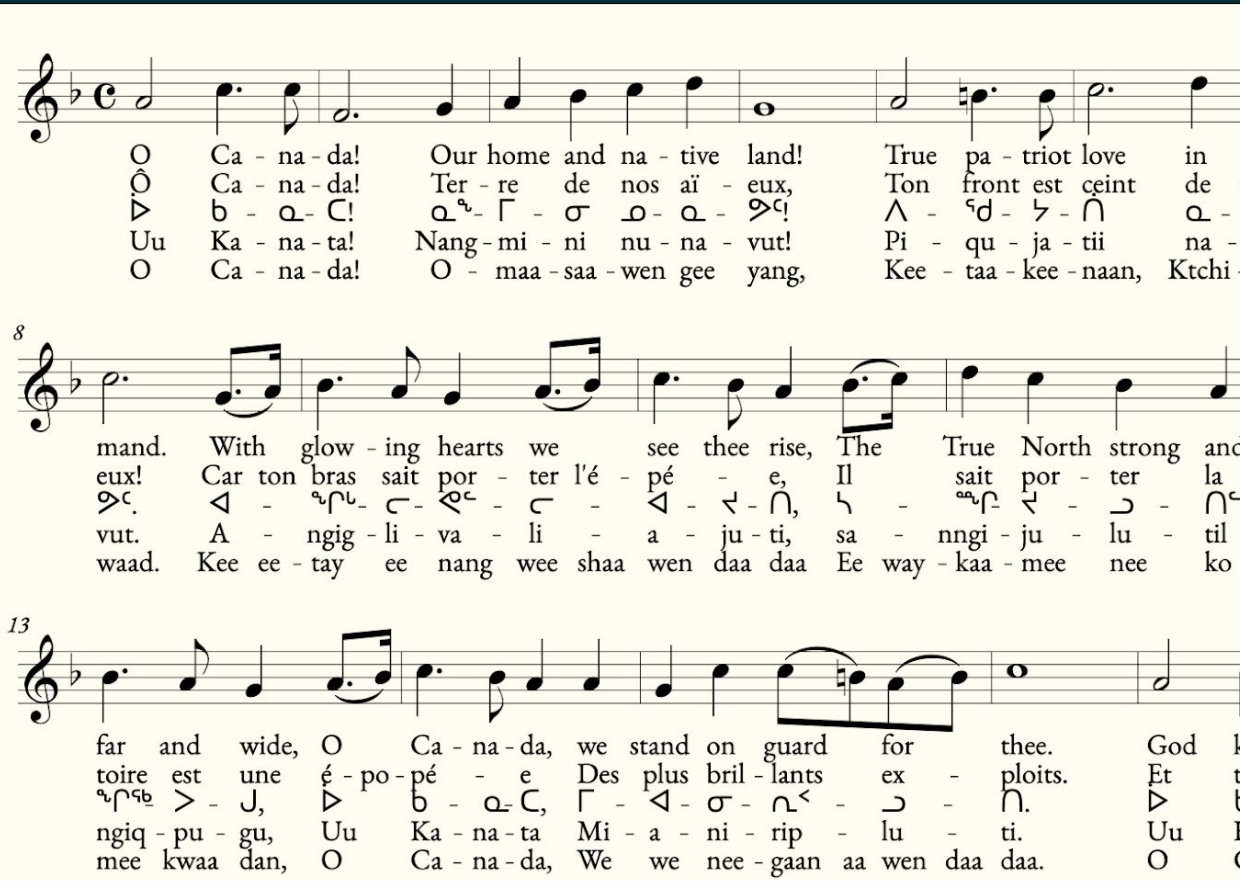


Verovio Humdrum Viewer (Website)

After trying various other methods such as *xml2hum* and *mid2hum*, we finally ended up using *Verovio Humdrum Viewer* for converting musicXML files to kern. As shown below, the text on the left (kern) is rendered as music, making it easier to visualize the text encoding.

VerovioHumdrumViewer Calixa Lavallée,

	**kern	**text	**text	**text
2	*clefG2	*	*	*
3	*k[b-]	*	*	*
4	*F:	*	*	*
5	*M4/4	*	*	*
6	*met(c)	*	*	*
7	*MM140	*	*	*
8	!	!ENG	!FRE	!IKU
9	=1-	=1-	=1-	=1-
10	2a	O	Ô	ᐅ
11	4.cc	Ca-	Ca-	ᑲ-
12	8cc	-na-	-na-	-ᓇ-
13	=2	=2	=2	=2
14	2.f	-da!	-da!	-ᑕ!
15	4g	Our	Ter-	ᓇᐧ-
16	=3	=3	=3	=3
17	4a	home	-re	-ᒥ-
18	4b-	and	de	-ᓂ
19	4cc	na-	nos	ᓄ-
20	4dd	-tive	aï-	-ᓇ-
21	=4	=4	=4	=4
22	1g	land!	-eux,	-ᐳᑦ!
23	=5	=5	=5	=5
24	2a	True	Ton	ᐱ-
25	4.bn	pa-	front	-ᖏd-
26	8b	-triot	est	-ᔭ-
27	=6	=6	=6	=6
28	2.cc	love	ceint	-ᑎ
29	4dd	in	de	ᓇ-
30	=7	=7	=7	=7
31	4ee	all	fleu-	-ᓪᑦ-
32	4ee	thy	-rons	-ᑎ-
33	4dd	sons	glo-	-ᐊᖅ-
34	4dd	com-	-ri-	-ᐸ-
35	=8	=8	=8	=8
36	2.cc	-mand.	-eux!	-ᐳᑦ.



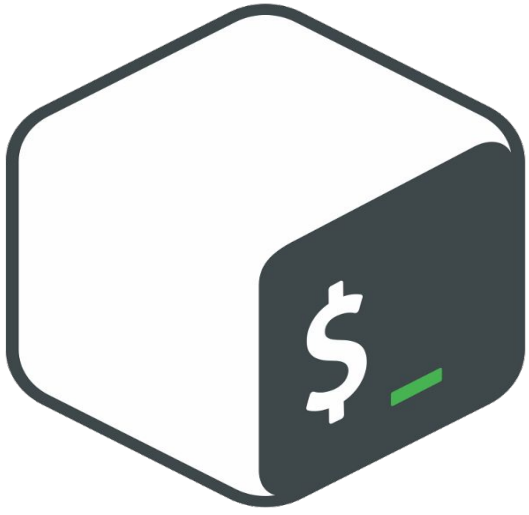
Part B:

Scripts and Programs

What follows is the code we wrote to extract data and information from the text (kern) files, as well as do some complex analysis and evaluation of equations.

This is technical code, included to show the vast amounts of programs and algorithms written for our project. The outputs of these were then put into a text file, and finally uploaded to our spreadsheet.

(The titles of each of these programs accurately describe the function of each of them.)



BASH

Average Note Value

```
#!/bin/bash

sumfile=/tmp/humdrumsum
file=$1

if [ -f $sumfile ]; then
   echo "Sumfile exists; instance of script may already be running; exiting..."
   exit 1
fi

a=$(grep -v '=' $file | grep -o '[[:digit:]]*' | grep .)
b=$(grep -v '=' $file | grep -o '[[:digit:]]*' | grep . | wc -l)

sum=0
grep -v '=' $file | grep -o '[[:digit:]]*' | grep . | while read line
do
   sum=$(($sum+$line))
   echo $sum > $sumfile
done

a=$(cat $sumfile) && rm $sumfile

c=$(bc -l <<< $a/$b)
echo 1/$c
```

Average Pitch

```
#!/bin/bash
file=$1

summedpitches="$(deg $file | grep -v '=' | grep -v '*' | grep -v '!' |
tr "\t" "~" | cut -d'~' -f1 | grep -v 'r' | tr -d ^ | tr -d v | tr -d .
| sed '/^$/d' | sed 's/+/.5/g' | sed 's/1-/0.5/g' | sed 's/2-/1.5/g' |
sed 's/3-/2.5/g' | sed 's/4-/3.5/g' | sed 's/5-/4.5/g' | sed
's/6-/5.5/g' | sed 's/7-/6.5/g' | tr '\n' '+' | awk '{print $0"0"}' |
bc -l)"

totallines="$(deg $file | grep -v '=' | grep -v '*' | grep -v '!' | tr
"\t" "~" | cut -d'~' -f1 | grep -v 'r' | wc -l)"


bc -l <<< $summedpitches/$totallines
```

Average Steps

```
#!/bin/bash
file=$1

fileprep=$(deg $file | grep '.' | grep -v '!' | grep -v '*' | grep -v '=' | grep -o
'[[:digit:]]*' | grep '.' | perl -lne 'if ($.==1){$p=$_} else{print "$p ".($_-$p); $p=$_}
END{print $p}' | tr ' ' '~' | cut -d '~' -f2)

divideBy=$( echo "$fileprep" | wc -l)

includingNegativesSum=$( echo "$fileprep" | tr '\n' '+' | awk '{print $0"0"}' | bc -l)

absoluteValueSum=$( echo "$fileprep" | tr -d '-'  | tr '\n' '+' | awk '{print $0"0"}' | bc -l)

absoluteValueEvaluate=$(bc -l <<< $absoluteValueSum/$divideBy)

includingNegativesEvaluate=$(bc -l <<< $includingNegativesSum/$divideBy)

echo $absoluteValueEvaluate
echo $includingNegativesEvaluate

lastLine=$( echo "$fileprep" | tail -n1 |  tr -d '-' )
firstline=$( echo "$fileprep" | head -n1 |  tr -d '-' )

bc -l <<< $firstline-$lastLine | tr -d '-'
```

Key Signature

```
#!/bin/bash
file=$1
a=$(grep 'k\[' $file)

#if $a='*k[b-]'
#then echo toaster

#else echo boi

if [ "$a" == "*k[]" ]; then
   echo C Major
elif [ "$a" == "*k[f#]" ]; then
   echo G Major
elif [ "$a" == "*k[f#c#]" ]; then
   echo D Major
elif [ "$a" == "*k[f#c#g#]" ]; then
   echo A Major
elif [ "$a" == "*k[f#c#g#d#]" ]; then
   echo E Major
elif [ "$a" == "*k[f#c#g#d#a#]" ]; then
   echo B Major
elif [ "$a" == "*k[f#c#g#d#a#e#]" ]; then
   echo F\# Major
elif [ "$a" == "*k[f#c#g#d#a#e#b#]" ]; then
   echo C\#  Major
elif [ "$a" == "*k[b-]" ]; then
   echo F Major
elif [ "$a" == "*k[b-e-]" ]; then
   echo Bb Major
elif [ "$a" == "*k[b-e-a-]" ]; then
   echo Eb Major
elif [ "$a" == "*k[b-e-a-d-]" ]; then
   echo Ab Major
elif [ "$a" == "*k[b-e-a-d-g-]" ]; then
   echo Db Major
elif [ "$a" == "*k[b-e-a-d-g-c-]" ]; then
   echo Gb Major
elif [ "$a" == "*k[b-e-a-d-g-c-f-]" ]; then
   echo Cb Major
fi
```

Most Used Note Value

```
file=$1
fileprep="$(grep -v '=' $file | grep -v '*' | grep -v
'!' |  grep -o '[[:digit:]]*' | grep '.')"

totallines=$(echo "$fileprep" | wc -l)
totaltime=$(echo "$fileprep" | sed 's|^|1/|' | tr '\n'
'+' | awk '{print $0"0"}' | bc -l)

one=$(echo "$fileprep" | grep '\b1\b' | wc -l)
onetime=$one
two=$(echo "$fileprep" | grep '\b2\b' | wc -l)
twotime=$( bc -l <<< $two/2)
four=$(echo "$fileprep" | grep '\b4\b' | wc -l)
fourtime=$( bc -l <<< $four/4)
eight=$(echo "$fileprep" | grep '\b8\b' | wc -l)
eighttime=$( bc -l <<< $eight/8)
sixteen=$(echo "$fileprep" | grep '\b16\b' | wc -l)
sixteentime=$( bc -l <<< $sixteen/16)
thirtytwo=$(echo "$fileprep" | grep '\b32\b' | wc -l)
thirtytwotime=$( bc -l <<< $thirtytwo/32)
sixtyfour=$(echo "$fileprep" | grep '\b64\b' | wc -l)
sixtyfourtime=$( bc -l <<< $sixtyfour/64)

array=($one $two $four $eight $sixteen $thirtytwo
$sixtyfour)
timearray=($onetime $twotime $fourtime $eighttime
$sixteentime $thirtytwotime $sixtyfourtime)
```

```
largest=$(printf '%s\n' "${array[@]}" |
sort -rn | sed '1q;d')

if [ "$largest" = $one ]
then
   greatest=1
elif [ "$largest" = $two ]
then
   greatest=2
elif [ "$largest" = $four ]
then
   greatest=4
elif [ "$largest" = $eight ]
then
   greatest=8
elif [ "$largest" = $sixteen ]
then
   greatest=16
elif [ "$largest" = $thirtytwo ]
then
   greatest=32
elif [ "$largest" = $sixtyfour ]
then
   greatest=64
fi

echo $greatest
bc -l <<< $largest/$totallines
```

Most Used Pitch

```
file=$1
fileprep=$(deg $file | grep -v '=' | grep -v '*' |
grep -v '!' | tr  "\t" "~" |  cut -d '~' -f1 )
totallines=$( echo "$fileprep" | grep -v '\.' |
grep -v 'r' | sed '/^$/d' | wc -l)

seven="$(echo "$fileprep" | grep '7' | grep -v
'7-' | grep -v '7+' | wc -l)"
sevenminus="$(echo "$fileprep" | grep '7-' | wc
-l)"
sevenplus="$(echo "$fileprep" | grep '7+' | wc
-l)"
six="$(echo "$fileprep" | grep '6' | grep -v '6-'
| grep -v '6+' | wc -l)"
sixminus="$(echo "$fileprep" | grep '6-' | wc -l)"
sixplus="$(echo "$fileprep" | grep '6+' | wc -l)"
five="$(echo "$fileprep" | grep '5' | grep -v '5-'
| grep -v '5+' | wc -l)"
fiveminus="$(echo "$fileprep" | grep '5-' | wc
-l)"
fiveplus="$(echo "$fileprep" | grep '5+' |  wc
-l)"
four="$(echo "$fileprep" | grep '4' | grep -v '4-'
| grep -v '4+' | wc -l)"
fourminus="$(echo "$fileprep" | grep '4-' | wc
-l)"
fourplus="$(echo "$fileprep" | grep '4+' | wc
-l)"
three="$(echo "$fileprep" | grep '3' | grep -v
'3-' | grep -v '3+' | wc -l)"threeplus="$(echo
"$fileprep" | grep '3+' | wc -l)"
threeminus="$(echo "$fileprep" | grep '3-' | wc
-l)"
two="$(echo "$fileprep" | grep '2' | grep -v
'2-' | grep -v '2+' | wc -l)twoplus="$(echo
"$fileprep" | grep '2+' | wc -l)"
twominus="$(echo "$fileprep" | grep '2-' | wc
-l)"
one="$(echo "$fileprep" | grep '1' | grep -v
'1-' |  grep -v '1+' |wc -l)"
oneplus="$(echo "$fileprep" | grep '1+' | wc
-l)"
oneminus="$(echo "$fileprep" | grep '1-' | wc
-l)"
onelowered=$oneminus
oneraised=$((oneplus+twominus))
tworaised=$((twoplus+threeminus))
threeraised=$((threeplus+fourminus))
fourraised=$((fourplus+fiveminus))
fiveraised=$((fiveplus+sixminus))
sixraised=$((sixplus+minus))
sevenraised=$sevenplus
```

Percent of Song In

```
file=$1

a=$(grep -v '=' $file | grep -v '*' | grep -v '!!' | grep '8' | grep -v '8\.'
| grep -v 'r' | wc -l) # isolate note, find count
c=$(bc -l <<< $a*'1/8') # multiply by note value
b=$(grep -v '=' $file | grep -o '[[:digit:]]*' | grep . | sed 's|^|1/|' | tr
'\n' '+' | awk '{print $0"0"}' | bc -l)

bc -l <<< $c/$b
```

Repeated Note Value

```
# #!/bin/bash
file=$1
rhythmfile=$(grep -v '*' $file | grep -v '=' | grep -v '!' | grep -o
'[[:digit:]]*' | grep '.')
test="$(echo "$rhythmfile" | uniq -c | awk '{print $2,$1}' | tr  " " "~" |  cut
-d '~' -f2 | grep -v '\b1\b' | grep -v '\b2\b')"
bothcolumns="$(echo "$rhythmfile" | uniq -c | awk '{print $2,$1}' )"

three=$(echo "$test" | grep '\b3\b' | wc -l)
four=$(echo "$test" | grep '\b4\b' | wc -l)
five=$(echo "$test" | grep '\b5\b' | wc -l)
six=$(echo "$test" | grep '\b6\b' | wc -l)
above=$(echo "$test" | grep -v '\b3\b' | grep -v '\b4\b' | grep -v '\b5\b' |
grep -v '\b6\b' | wc -l)
sorted="$(echo "$test" | sort -n)"
largest="${sorted##*$'\n'}"
valueoflargest="$(echo "$bothcolumns" | grep "${sorted##*$'\n'}" | tr  " " "~"
|  cut -d '~' -f1 | sort -n | head -n 1 | sed 's|^|1/|')"
echo $three
echo $four
echo $five
echo $above
echo $largest
bc -l <<< $valueoflargest*$largest
```

Repeated Pitch

```
# #!/bin/bash
file=$1
rhythmfile=$(grep -v '*' $file | grep -v '=' | grep -v '!' | grep -o '[[:digit:]]*'
| grep '.')
test="$(echo "$rhythmfile" | uniq -c | awk '{print $2,$1}' | tr  " " "~" |  cut -d
'~' -f2 | grep -v '\b1\b' | grep -v '\b2\b')"
bothcolumns="$(echo "$rhythmfile" | uniq -c | awk '{print $2,$1}' )"

three=$(echo "$test" | grep '\b3\b' | wc -l)
four=$(echo "$test" | grep '\b4\b' | wc -l)
five=$(echo "$test" | grep '\b5\b' | wc -l)
six=$(echo "$test" | grep '\b6\b' | wc -l)
above=$(echo "$test" | grep -v '\b3\b' | grep -v '\b4\b' | grep -v '\b5\b' | grep
-v '\b6\b' | wc -l)
sorted="$(echo "$test" | sort -n)"
largest="${sorted##*$'\n'}"
valueoflargest="$(echo "$bothcolumns" | grep "${sorted##*$'\n'}" | tr  " " "~" |
cut -d '~' -f1 | sort -n | head -n 1
```

Rhythmic Themes

```
# #!/bin/bash

count() {
   tailnumber=$(( $3 - $2 + 1 ))
   pattern=$(echo  "$1" | head -n $3 | tail -n $tailnumber)
   echo "$1" | pcregrep -Mc "^\Q$(echo "$pattern")\E$"
}


file=$1
fileprep=$(grep -v '=' $file | grep -v '!' | grep -v '*' |  grep -o '[[:digit:]]*' | grep . )

linecount=$(echo "$fileprep" | wc -l)
start=1


for i in {2..45}; do
   len=$i
   end=$(( $linecount - $len + 1 ))
```

Rhythmic Themes (Cont.)

```
    test=
    for j in $(seq $start $end); do
        test="$test\n$(count "$fileprep" $j $((j+len-1)))"
    done

    a=$(printf $test | grep -v '\b1\b' )
    totalrepetitions=$(echo "$a" | wc -l)
    mostrepetitions=$(echo "$a" | sort -rn | head -n1)
    mostrepetitions=$(echo "$a" | sort -rn | head -n1)

    var2=
    for k in $(seq 1 $mostrepetitions); do
        var1=$(printf "$a" | grep '\b'$k'\b' | wc -l)
        var2="$var2\n$(echo $(( var1 / k )))"
    done
    printf "$var2" | tr '\n' '+' | awk '{print "0"$0}' | bc -l
    echo $mostrepetitions
    echo $totalrepetitions
done
```

Scales

```
#!/bin/bash
file=$1

ascendingSingle=$(deg $file | grep -v '!' | grep -v '=' | grep -v '*' | tr  "\t" "~" |  cut -d '~' -f
| tr '.' 'r' | sed 's/^.//' | grep '.' | grep -v '-' | grep -v '+' | awk
'{l=p=$1}{while((r=getline)>=0){if($1==p+1){p=$1;continue};print(l==p?l:l","p);l=p=$1;if(r==0){ break
};}}' |  grep ',' | tr ',' '\t' | awk 'BEGIN { OFS = "\t" } { $3 = $2 - $1 } 1' | tr "\t" "~" | cut -
'~' -f3)
descendingSingle=$(deg $file | grep -v '!' | grep -v '=' | grep -v '*' | tr  "\t" "~" |  cut -d '~' -
| tr '.' 'r' | sed 's/^.//' | grep '.' | grep -v '-' | grep -v '+' | awk
'{l=p=$1}{while((r=getline)>=0){if($1==p-1){p=$1;continue};print(l==p?l:l","p);l=p=$1;if(r==0){ break
};}}' |  grep ',' | tr ',' '\t' | awk 'BEGIN { OFS = "\t" } { $3 = $2 - $1 } 1' | tr "\t" "~" | cut -
'~' -f3)
ascendingDouble=$(deg $file | grep -v '!' | grep -v '=' | grep -v '*' | tr  "\t" "~" |  cut -d '~' -f
| tr '.' 'r' | sed 's/^.//' | grep '.' | grep -v '-' | grep -v '+' | awk
'{l=p=$1}{while((r=getline)>=0){if($1==p+2){p=$1;continue};print(l==p?l:l","p);l=p=$1;if(r==0){ break
};}}' |  grep ',' | tr ',' '\t' | awk 'BEGIN { OFS = "\t" } { $3 = $2 - $1 } 1' |  tr "\t" "~" | cut
'~' -f3)
descendingDouble=$(deg $file | grep -v '!' | grep -v '=' | grep -v '*' | tr  "\t" "~" |  cut -d '~' -
| tr '.' 'r' | sed 's/^.//' | grep '.' | grep -v '-' | grep -v '+' | awk
'{l=p=$1}{while((r=getline)>=0){if($1==p-2){p=$1;continue};print(l==p?l:l","p);l=p=$1;if(r==0){ break
};}}' |  grep ',' | tr ',' '\t' | awk 'BEGIN { OFS = "\t" } { $3 = $2 - $1 } 1' | tr "\t" "~" | cut -
'~' -f3)

ascSingle=$(echo "$ascendingSingle" | while read i; do echo "$i+1" | bc; done )
descSingle=$(echo "$descendingSingle" | while read i; do echo "$i-1" | bc; done | sed 's/^.//')
descDouble=$(echo "$descendingDouble" | while read i; do echo "$i/2" | bc; done | while read i; do ec
"$i-1" | bc; done | sed 's/^.//')
ascDouble=$(echo "$ascendingDouble" | while read i; do echo "$i/2" | bc; done | while read i; do echo
"$i+1" | bc; done )
```

Scales (Cont.)

```
ascDoubleTwo=$( echo "$ascDouble" | grep '\b2\b' | wc -l)
ascDoubleThree=$( echo "$ascDouble" | grep '\b3\b' | wc -l)
ascDoubleFour=$( echo "$ascDouble" | grep '\b4\b' | wc -l)
ascDoubleFive=$( echo "$ascDouble" | grep '\b5\b' | wc -l)
ascDoubleSix=$( echo "$ascDouble" | grep '\b6\b' | wc -l)
ascDoubleAbove=$( echo "$ascDouble" | grep -v '\b6\b' | grep -v '\b5\b' | grep -v '\b4\b
| grep -v '\b3\b' | grep -v '\b2\b' | wc -l)
ascDoubleLargest=$( echo "$ascDouble" | sort -n |  tail -n1 )

descSingleTwo=$( echo "$descSingle" | grep '\b2\b' | wc -l)
descSingleThree=$( echo "$descSingle" | grep '\b3\b' | wc -l)
descSingleFour=$( echo "$descSingle" | grep '\b4\b' | wc -l)
descSingleFive=$( echo "$descSingle" | grep '\b5\b' | wc -l)
descSingleSix=$( echo "$descSingle" | grep '\b6\b' | wc -l)
descSingleAbove=$( echo "$descSingle" | grep -v '\b6\b' | grep -v '\b5\b' | grep -v
'\b4\b' | grep -v '\b3\b' | grep -v '\b2\b' | wc -l)
descSingleLargest=$( echo "$descSingle" | sort -n |  tail -n1 )

descDoubleTwo=$( echo "$descDouble" | grep '\b2\b' | wc -l)
descDoubleThree=$( echo "$descDouble" | grep '\b3\b' | wc -l)
descDoubleFour=$( echo "$descDouble" | grep '\b4\b' | wc -l)
descDoubleFive=$( echo "$descDouble" | grep '\b5\b' | wc -l)
descDoubleSix=$( echo "$descDouble" | grep '\b6\b' | wc -l)
descDoubleAbove=$( echo "$descDouble" | grep -v '\b6\b' | grep -v '\b5\b' | grep -v
'\b4\b' | grep -v '\b3\b' | grep -v '\b2\b' | wc -l)
```

Formula Evaluation

```
#!/bin/bash

songFile=$1

periodFile=$2

rFile=$3

rTotal=$( cat $rFile | tr -d '-' |
awk '{T+=$0} END { printf "%.2f\n",
T }' )

start=1
end=$(cat "$rFile" | wc -l)

for i in $(seq $start $end); do
   rNumbers=$(cat $rFile | tr -d
'-' | sed "${i}q;d" )
   test="$test$(echo
"$rNumbers/$rTotal" | bc -l )\n"
done
```

```
for j in $(seq $start $end); do

   songNumber=$(cat $songFile | sed "${j}q;d" )
   periodNumber=$(cat $periodFile | sed "${j}q;d" )
   difference=$(echo "$songNumber-$periodNumber" | bc -l | tr
   multiplier=$(echo "$periodNumber*$songNumber" | bc -l)
   squareRoot=$(echo "scale=2;sqrt($multiplier)" | bc)
   divider=$(echo "$difference/$squareRoot" | bc -l)
   underono="$underono$(echo "1-$divider" | bc -l)\n"

Done

underone=$(printf "/$underono/" | tr -d '/' | tr ' ' '\n')

for h in $(seq $start $end); do
   ratioline=$(echo "$underone" | sed "${h}q;d")
   weightline=$(printf "$test" | sed "${h}q;d")
   multiplied="$multiplied$(echo "$ratioline*$weightline" | bc

done

###############    Above 'for' loop multiplies all coefficient
their corresponding ratio

result=$(printf "/$multiplied/" | tr -d '/' | grep . | tr '\n'
awk '{print $0"0"}' | bc -l)

echo "Percent correlation: $result"
```

Melody Isolation

```
#!/bin/bash

songFile=$1

periodFile=$2

rFile=$3

rTotal=$( cat $rFile | tr -d '-' | awk
'{T+=$0} END { printf "%.2f\n", T }' )

start=1
end=$(cat "$rFile" | wc -l)

for i in $(seq $start $end); do
   rNumbers=$(cat $rFile | tr -d '-' |
sed "${i}q;d" )
   test="$test$(echo "$rNumbers/$rTotal"
| bc -l )\n"
done
```

```
for j in $(seq $start $end); do

   songNumber=$(cat $songFile | sed
"${j}q;d" )
   periodNumber=$(cat $periodFile | sed
"${j}q;d" )
   difference=$(echo
"$songNumber-$periodNumber" | bc -l | tr
-d '-' )
   multiplier=$(echo
"$periodNumber*$songNumber" | bc -l)
   squareRoot=$(echo
"scale=2;sqrt($multiplier)" | bc)
   divider=$(echo
"$difference/$squareRoot" | bc -l)
   underono="$underono$(echo "1-$divider"
| bc -l)\n"

done

underone=$(printf "/$underono/" | tr -d
```

Automation of Formula Evaluation

```
for i in $( ls $directory); do renaissance=$(bash /Path/to/equation/script.sh
$directory/$i /Path/to/renaissance.txt /Path/to/R_Values.txt ); classical=$(bash
/Path/to/equation/script.sh $directory/$i /Path/to/classical.txt
/Path/to/R_Values.txt ); baroque=$(bash /Path/to/equation/script.sh $directory/$i
/Path/to/baroque.txt /Path/to/R_Values.txt ); romantic=$(bash
/Path/to/equation/script.sh $directory/$i /Path/to/romantic.txt
/Path/to/R_Values.txt ); twenty=$(bash /Path/to/equation/script.sh $directory/$i
/Path/to/20th_century.txt /Path/to/R_Values.txt ); && echo "Renaissance:
$renaissance" > $directory/$i_outputs.txt && echo "Baroque: $baroque" >>
$directory/$i_outputs.txt && echo "Classical: $classical" >>
$directory/$i_outputs.txt && echo "Romantic: $romantic" >>
$directory/$i_outputs.txt && echo "20th: $twenty" >> $directory/$i_outputs.txt;
done
```

Automation of Formula Evaluation (Cont.)

```
#!/bin/bash

directory=$1

rpath=/Users/svernooy/Desktop/EquationTester/rValues.txt

len=$(ls "$directory" | tr '\t' '\n' | wc -l)
lenless=$(echo $len-2 | bc -l)

argone=
argtwo=
```

```
for i in $(seq 0 $lenless); do
   currentnumber=$(echo $len-$i | bc -l)
   currentdirec=$(ls $directory | tail -n $currentnumber)

   for b in $(seq 2 $currentnumber); do
       argone=$(echo "$currentdirec" | sed '1q;d')
       argtwo=$(echo "$currentdirec" | sed "${b}q;d")
       echo $argone
       echo $argtwo
       if [ "$argone" = "$argtwo" ]; then
           echo ""
       else
           bash /Users/svernooy/Desktop/equation_geo.sh $directory/$argone $directory/$argtwo $rpath
       fi

       printf "\n"
       printf "\n"
   done
```


python

Formula Evaluation

```
def calculate_correlation(a_file, b_file):
    import os
    from scipy.stats.mstats import gmean
    a_values = []
    b_values = []
    r_values = []
    setname1 = []
    setname2 = []
    with open(a_file) as file:
        a = file.read().split('\n')
        aValues = [float(i) for i in a]
        for i in range(0, len(aValues)):
            if aValues[i] == 0:
                a_values.append(0.000000000000001)
            else:
                a_values.append(aValues[i])
        setname1.append(os.path.basename(file.name))
    with open(b_file) as file:
        b = file.read().split('\n')
        bValues = [float(i) for i in b]
        for i in range(0, len(bValues)):
            if bValues[i] == 0:
                b_values.append(0.000000000000001)
            else:
                b_values.append(bValues[i])
        setname2.append(os.path.basename(file.name))
    setname_1, ext1 = os.path.splitext(setname1[0])
    setname_2, ext2 = os.path.splitext(setname2[0])
```

Formula Evaluation (Cont.)

```
with open('r_values.txt') as file:
    r = file.read().split('\n')
    rValues = [float(i) for i in r]
    for i in range(0, len(rValues)):
        r_values.append(abs(rValues[i]))
if len(r_values) != len(a_values):
    print('make sure your files have the same amount of data!')
elif len(r_values) != len(b_values):
    print('make sure your files have the same amount of data!')
elif len(a_values) != len(b_values):
    print('make sure your files have the same amount of data!')
else:
    rs = []
    zs = []
    values = []
    final_sum = []
    for x in range(0, len(r_values)):
        rs.append(r_values[x]/sum(r_values))
    for x in range(0, len(a_values)):
        dist = abs(a_values[x]-b_values[x])
        geo_mean = gmean([a_values[x], b_values[x]])
        zs.append(1-(dist/geo_mean))
    for x in range(0, len(zs)):
        values.append(zs[x]*rs[x])
    final_sum.append(sum(values))
    with open('correlation.txt', 'w+') as file:
        file.write('%s;%s;%s' % (setname_1, setname_2, final_sum[0]))
        file.write('\n')
```

Occurrences

```
f = open('filepath.txt')
sigs = f.read().split('\n')
f.close()
output = []
proportions = []
for x in sigs:
    if x not in output:
        output.append(x)
def occurrences():
    for x in range(0, len(output)):
        count = sigs.count(output[x])
        occurrence = "%s; %s" % (output[x], count)
        proportions.append(occurrence)
occurrences()
with open('filepath.txt', 'w+') as file:
    for x in range(0, len(proportions)):
        file.write(proportions[x])
        file.write('\n')
```

A value finder for Zipf's law

```
f = open('filepath.txt')
oldys = f.read().split('\n')
f.close()
length = len(oldys)
xs = []
for x in range(1, length+1):
    xs.append(x)
ys = [float(i) for i in oldys]
distances = []
def distance(a, xlist, ylist):
    difs = []
    for x in range(0, len(ylist)):
        n = ylist[0]
        y = n/(xlist[x])**a
        dif = abs(ylist[x]-y)
        difs.append(dif)
    distances.append(sum(difs)/len(difs))
for x in range(0, 2000):
    a = x/1000
    distance(a, xs, ys)
smallest_distance = min(i for i in distances)
a = (distances.index(smallest_distance))/1000
print(a)
```

Section 2:

Analysis

Coloring

Beats in Bytes uses a specific color scheme for each composer and time period as seen right.

This allows us to visually organize and represent our data in a consistent way. You will see this coloring consistently repeated over and over again in our graphs, charts, and paragraphs.

We will also analyze the entirety of classical music, which will generally (though not always) be shown in black.

Renaissance	Byrd
Renaissance	Josquin
Renaissance	Palestrina
Renaissance	Lasso
Renaissance	Monteverdi
Baroque	Bach
Baroque	Handel
Baroque	Pachelbel
Baroque	Telemann
Classical	Beethoven
Classical	Mozart
Classical	Haydn
Classical	Schubert
Classical	Clementi
Romantic	Chopin
Romantic	Brahms
Romantic	Tchaikovsky
Romantic	Schumann
Romantic	Mendelssohn
20th Century	Holst
20th Century	Sibelius
20th Century	Stravinsky
20th Century	Debussy
20th Century	Prokofiev

Key Signature

A key signature defines the pitches used in a song. Pitch is how high or low a note is. Each song has at least one key signature. We started with a pie chart of all songs to see how many songs were in each key signature. We found that 5 key signatures made up over 75% of songs, with the most common being C major, F major, G major, and D major, Bb major, Eb major, which are all very common key signatures.

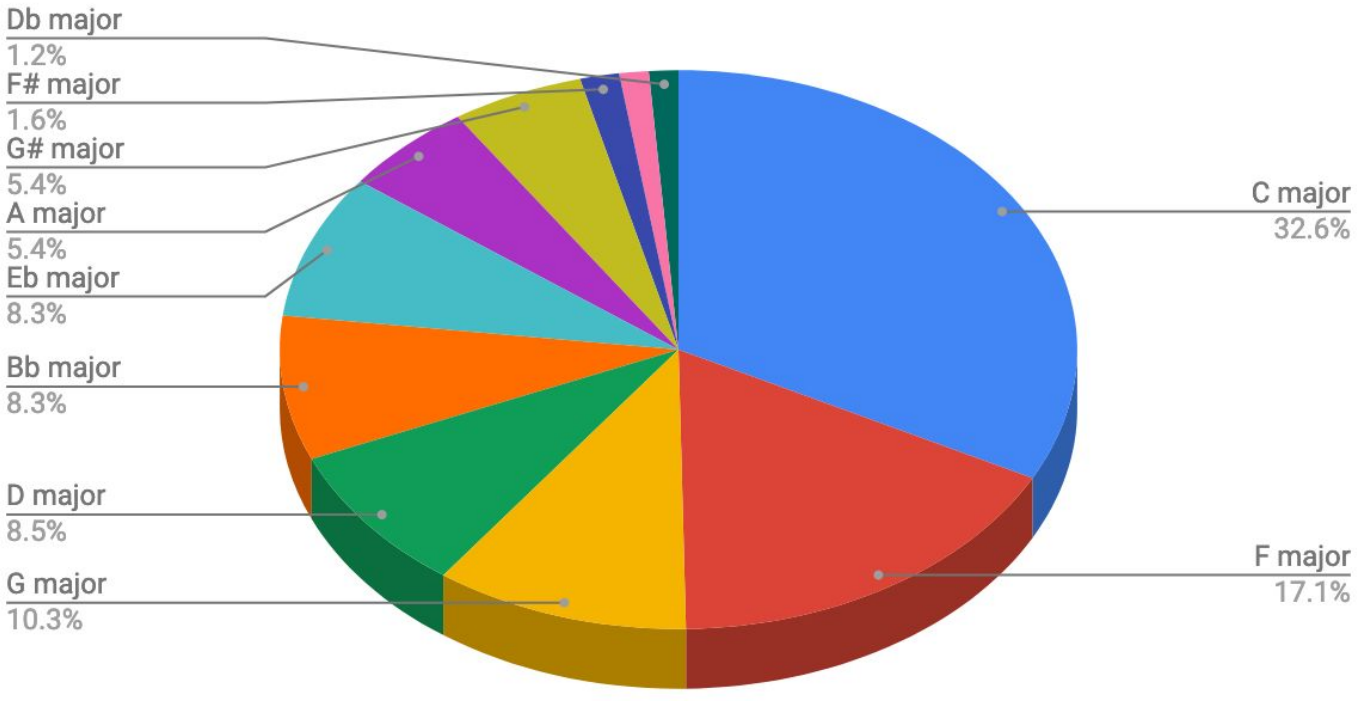


Fig. 1: A pie chart shows the percentage of songs with which key signature. C major and F major make up almost half of the songs (49.7%), with G major and D major also quite common. This is most likely due to the fact that C and F are the most simple key signatures, with no sharps and only one flat in the case of F major, and that G and D major contain only 1 and 2 sharps, respectively. It's very interesting how much more common Bb major, Eb major, and to a lesser extent, G# major were compared to what we originally expected.

Key Signature (Cont.)

When analyzing key signature by time period, we ordered the key signatures by most common to least common for each time period to analyze for Zipf's law and other mathematical distributions. We then compared with Zipf's Law by finding the curves of best fit for each time period, which are presented below.

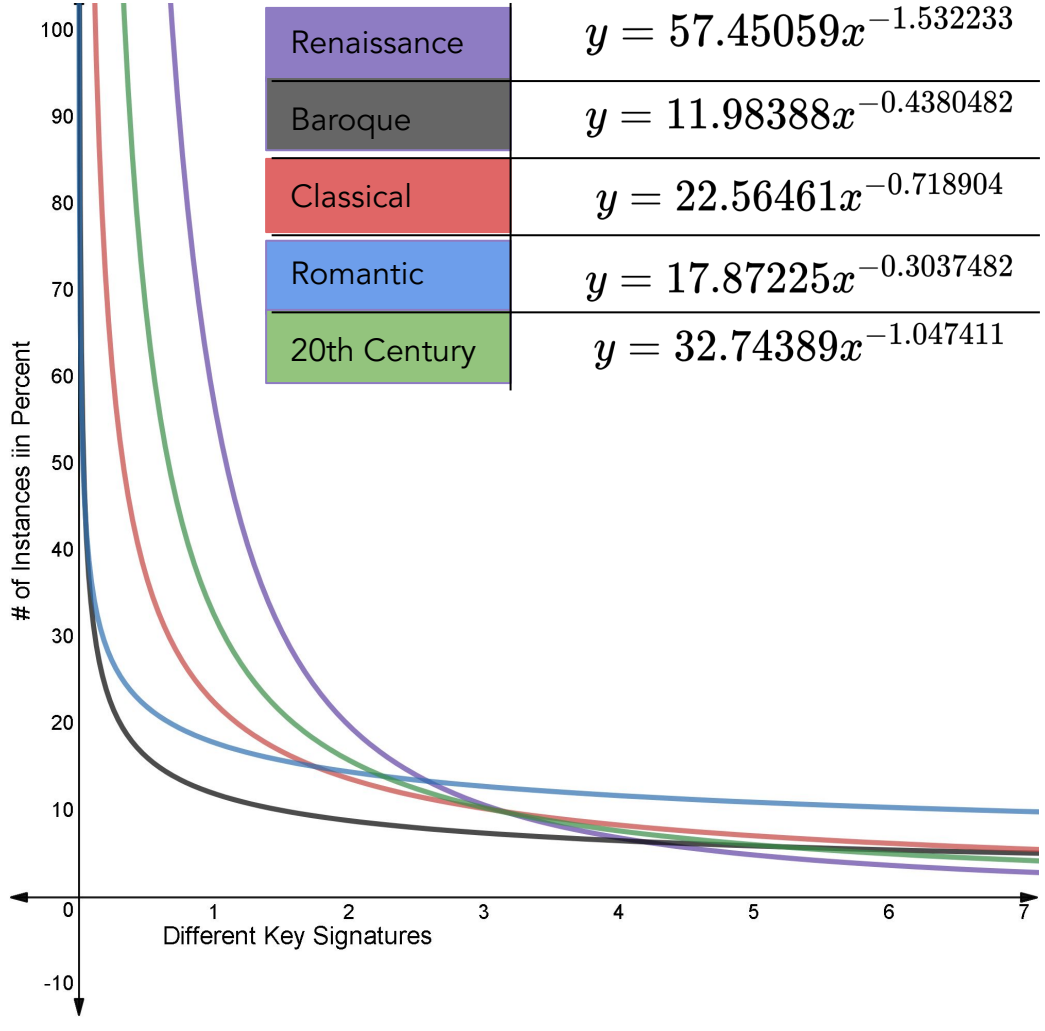


Fig. 2: Graphing the best fit equations reveals some interesting things. Renaissance has a steeper curve than any other time period (which is reflected across many metrics, as music then was very limited). Another point of note is Romantic, which has the flattest curve, and thus has a abnormally high number of less frequent key signatures represented in its music. However, the time period with the least songs in its most frequent key signature was baroque, which is somewhat surprising as although music was being experimented on, there was more variety as compared to both earlier and later periods, such as Classical or 20th Century.

Key Signature (Cont.)

Upon seeing the long tail, we instantly thought of other distribution laws. One such law that came to mind was Benford's law, famous for describing occurrences of specific numbers. After finding the values that Benford's law predicted, we plotted this against our own values. When plotted, the graph finds a relative closeness, that, while not exact, may show correlation to some extent. Doing a t-test on the values gives a p-value of 0.27546, thus the null hypothesis (that they are the same) cannot be rejected. What is also surprising is that the most common key signature, C major, shows up in exactly the same percentage of songs predicted by Benford's Law.

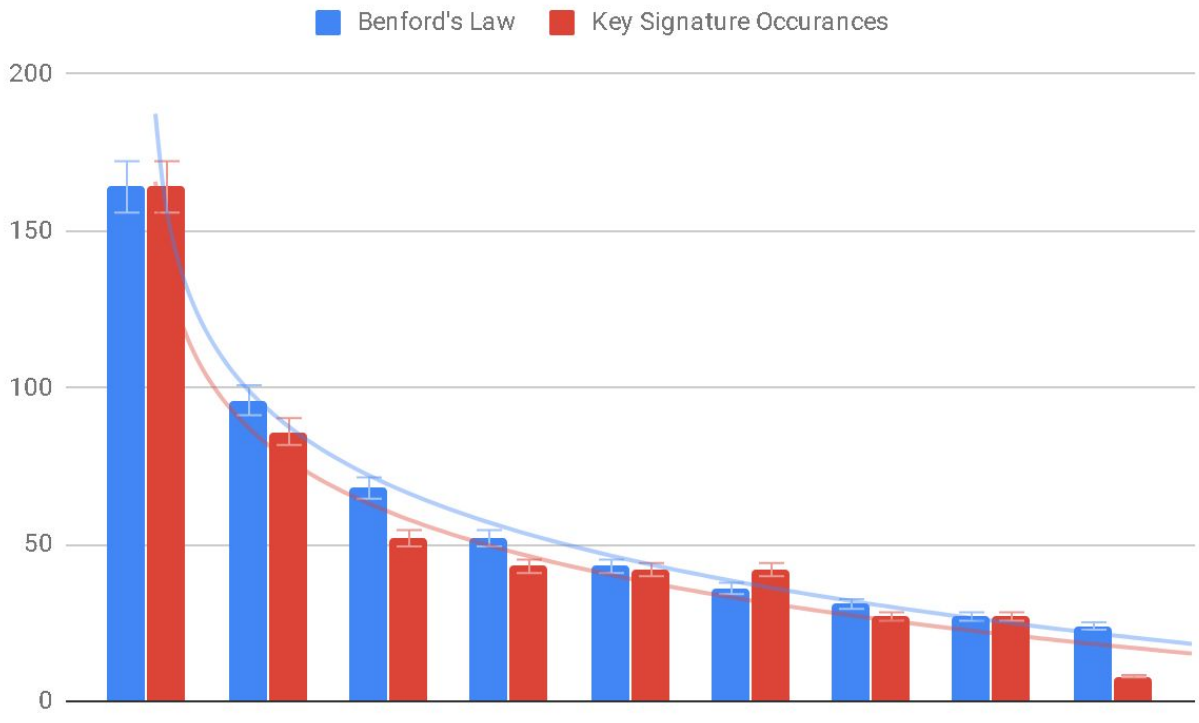


Fig. 3: A column chart shows the actual number of occurrences of a key signature arranged in descending order versus the number predicted by Benford's Law. With the exception of the 5th most common key signature, Benford's Law consistently has a higher percentage, which is caused in part by the flatter distribution of the occurences. Compared to Benford's Law, the second, third, and fourth most common key signatures (F major, G major, and D major, respectively) are less common.

Time Signature

Time signature defines certain aspect of a song's beats, and each song has at least one time signature. For the analysis of time signature, we were limited to occurrence-related measurements, due to non-numerical format of the data.
We began with mode, using a custom google sheets function, shown below:

```
=index(R1:R2, match(max(countif(R1:R2, R1:R2)), countif(R1:R2, R1:R2), 0))
```

The main other analysis opportunity was in occurrences. We used the occurrences python script to find this. This pie chart of Renaissance accurately depicts the dominance of a few time signatures, but there is still a large amount of diversity.

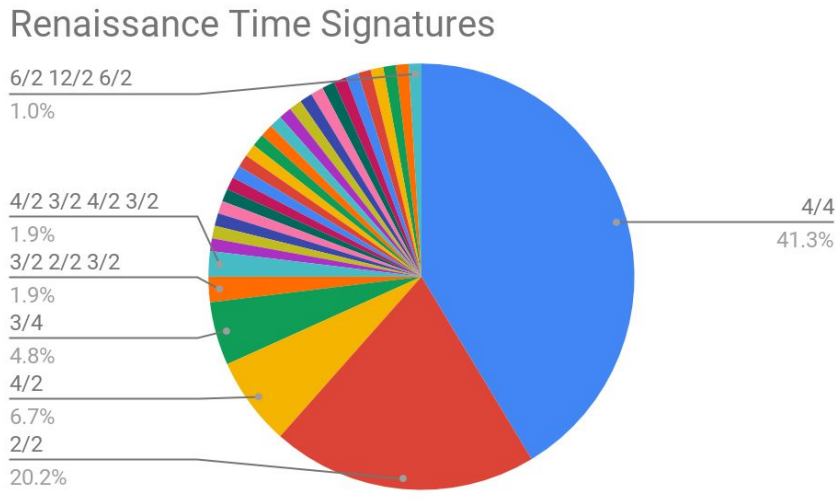


Fig 1. We found that the Renaissance time period used 2/2 time extremely frequently - exactly 41.3 percent of their songs. You can see how just 5 time signatures make up nearly 75% of songs (and we counted songs where the time signatures changed as different time signatures, hence the 3/2 2/2 3/2 seen here.)

Time Signature (Cont.)

We also attempted to find distribution laws in time signature. We began by graphing the best fit curve for our data compared to the values that were predicted by Zipf's law. The result showed only minor correlation, with an *a* value of 1.279.

As for Pareto distribution, it seemed to be present, but not exaggerated to the level of the 80/20 rule. Only about 65% of the songs in 2/2 time were from the Renaissance (which is 20% of our data).

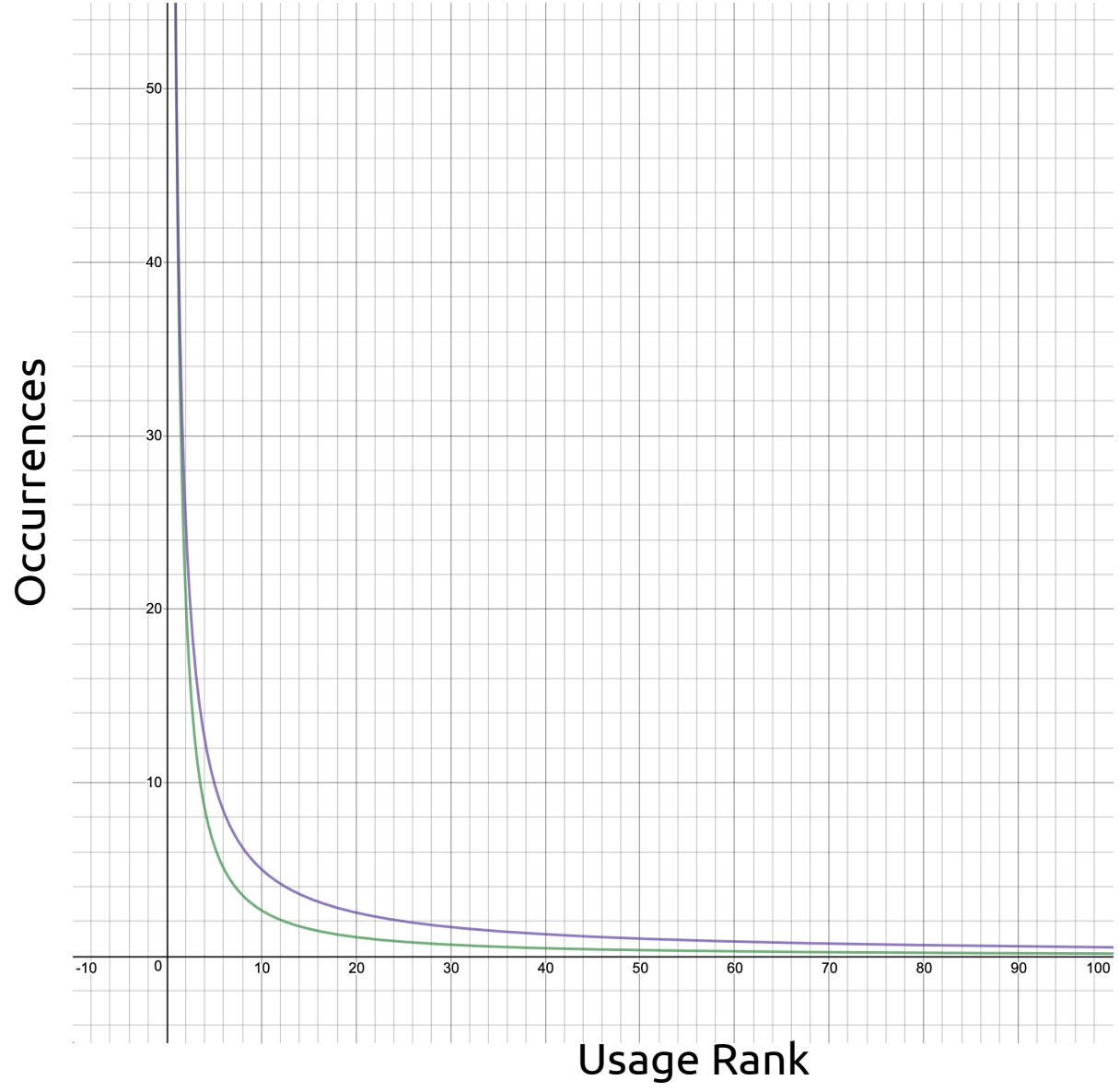


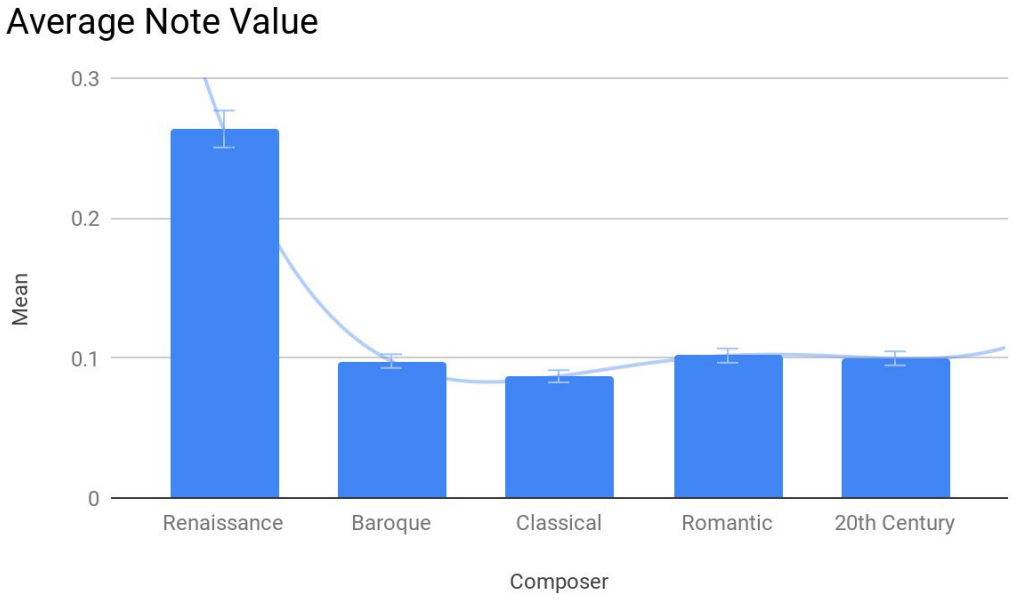
Fig 2: This graph shows the curve of the occurrences of our data (green) and that of Zipf's Law (purple).

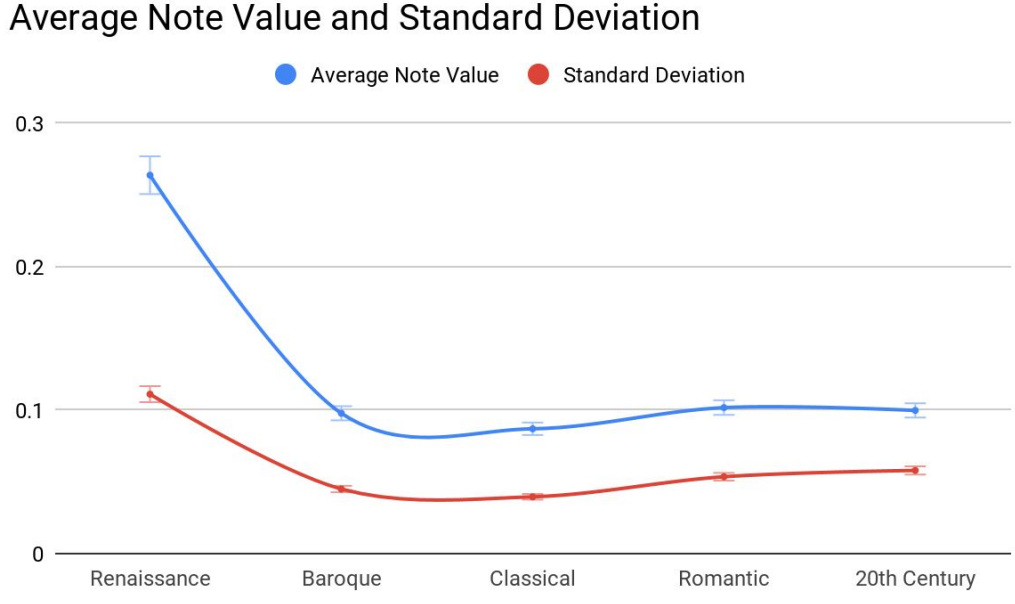
Average Note Value

Average note value is simply the average of all the different lengths of notes used in a song. In this case, we measured it by how much of a measure an average note takes up.

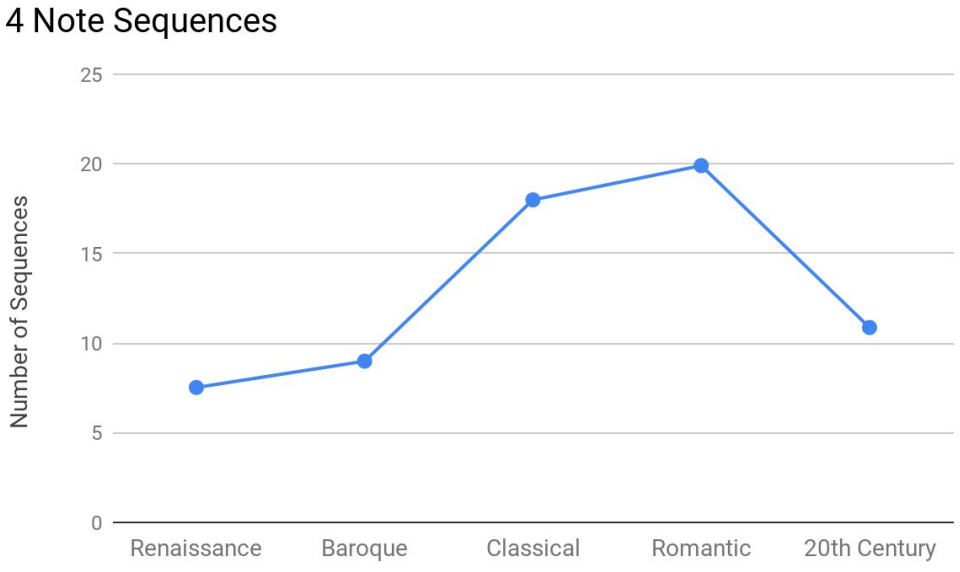
We noticed that composers tended to use much longer notes during the Renaissance than during any other time period, and that Classical had the shortest notes on average (all those 16th notes, probably)

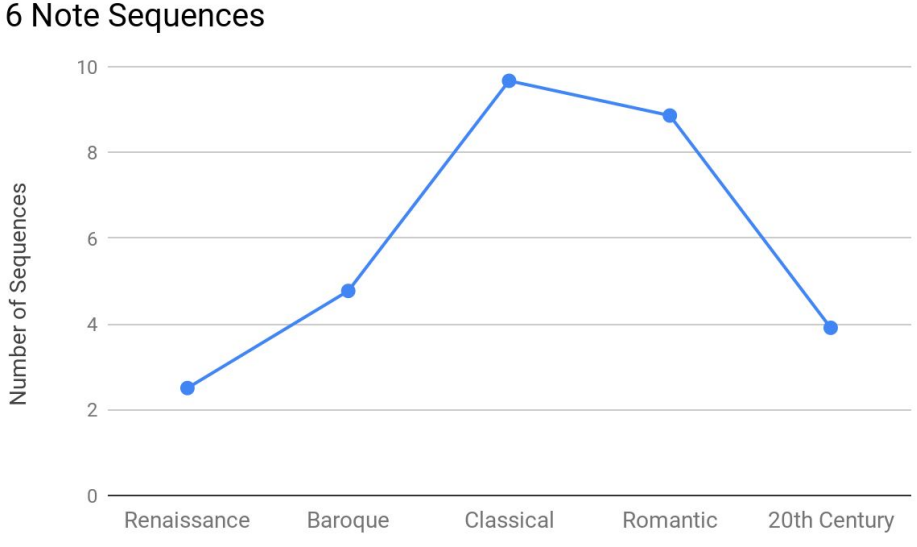
More interestingly, we learned that standard deviation creates an almost parallel curve to average note value. What this means is that as composers use longer notes, they are more likely to have greater variation in the length of their notes.





Sequences of Repeated Note Values

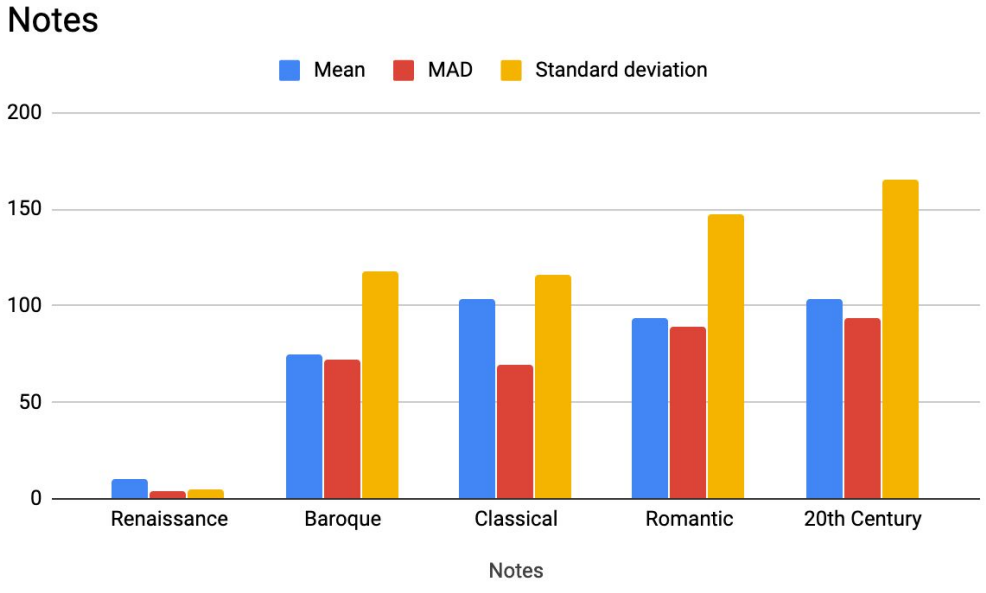


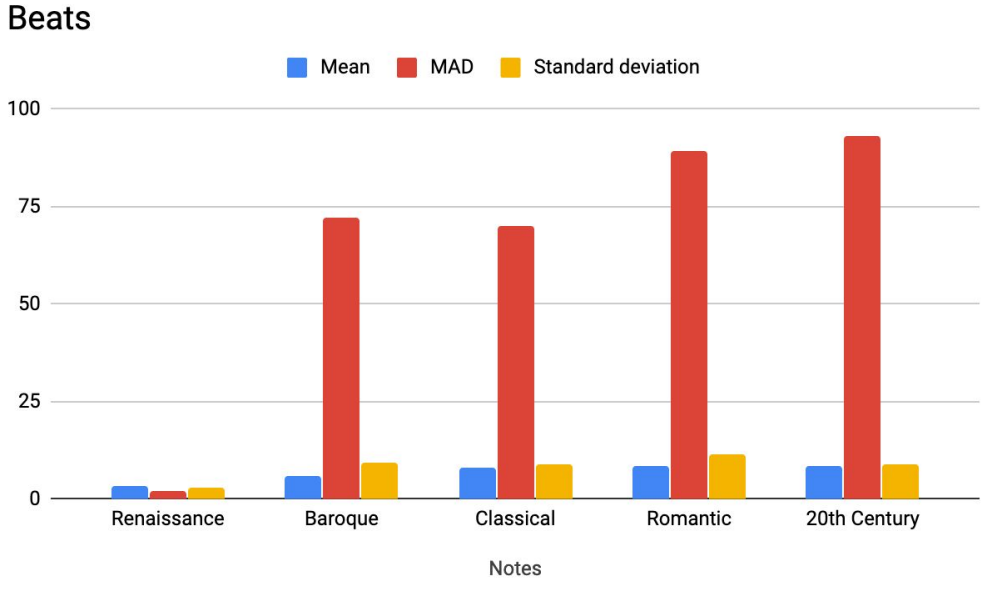


A sequence of repeated note values is a series of notes of the same length.

The number of sequences of repeated note values stayed consistent throughout all lengths for each time period. For all of them, the number of these sequences begins extremely low during the Renaissance, then slowly increases. Eventually, it drops back down for 20th Century. The peaks for the shorter sequences were in the Romantic time period, but after the sequences are greater than 5 notes long, Classical surpasses Romantic.

Sequences of Repeated Note Values (Cont.)



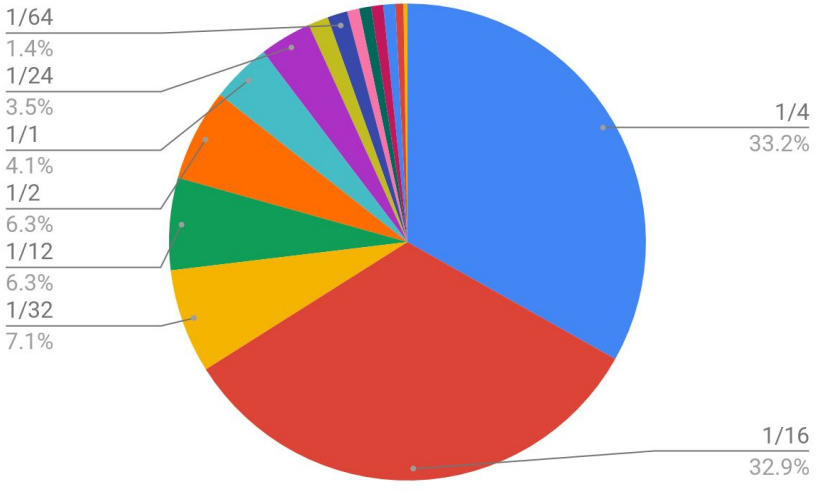


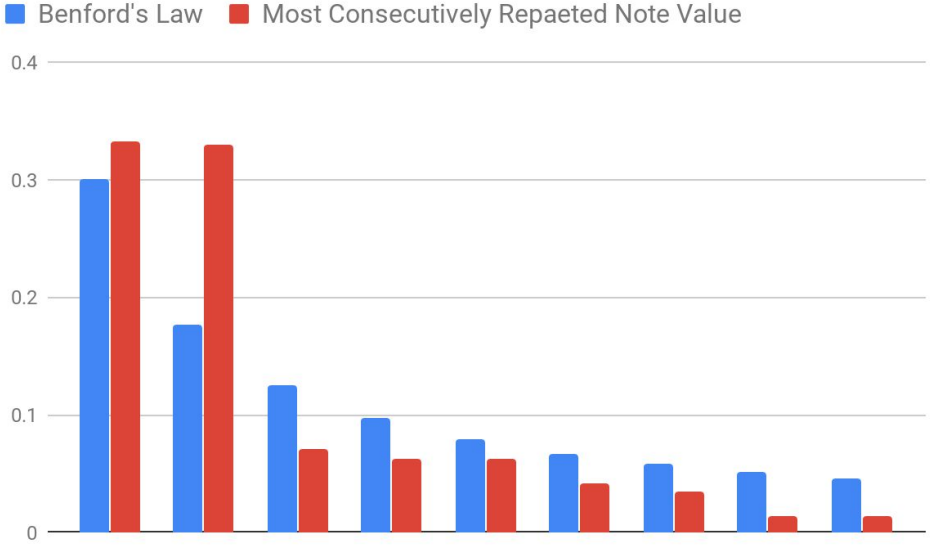
The length of the longest of these sequences was measured in both beats and notes. The only difference between the two is that measuring in beats considers the length of each note. The graphs show that Renaissance's longest sequences are much shorter than the other time periods, for both notes and beats. Also, when measuring in beats the mean absolute deviation (MAD) increases dramatically in the four later time periods, while staying no different in Renaissance.

Sequences of Repeated Note Values (Cont.)

We also conducted t-Tests between the length of longest in beats and in notes. We found that the two have hardly any correlation, because all of our *p* values were equal to or extremely close to 0.

We also found the most consecutively repeated note values. We measured occurrences, and found that the data was dominated by quarter notes and sixteenth notes. The data also was also comparable to Benford's Law.





Rhythmic Themes

A theme is a musical phrase (or sequence of notes) that is repeated 2 or more times throughout the song. In our songs, we looked for the amount of 2 notes long themes through 45 note long themes.

We then were able to graph these, and compare each time period graph. This gave us distinct results right away.

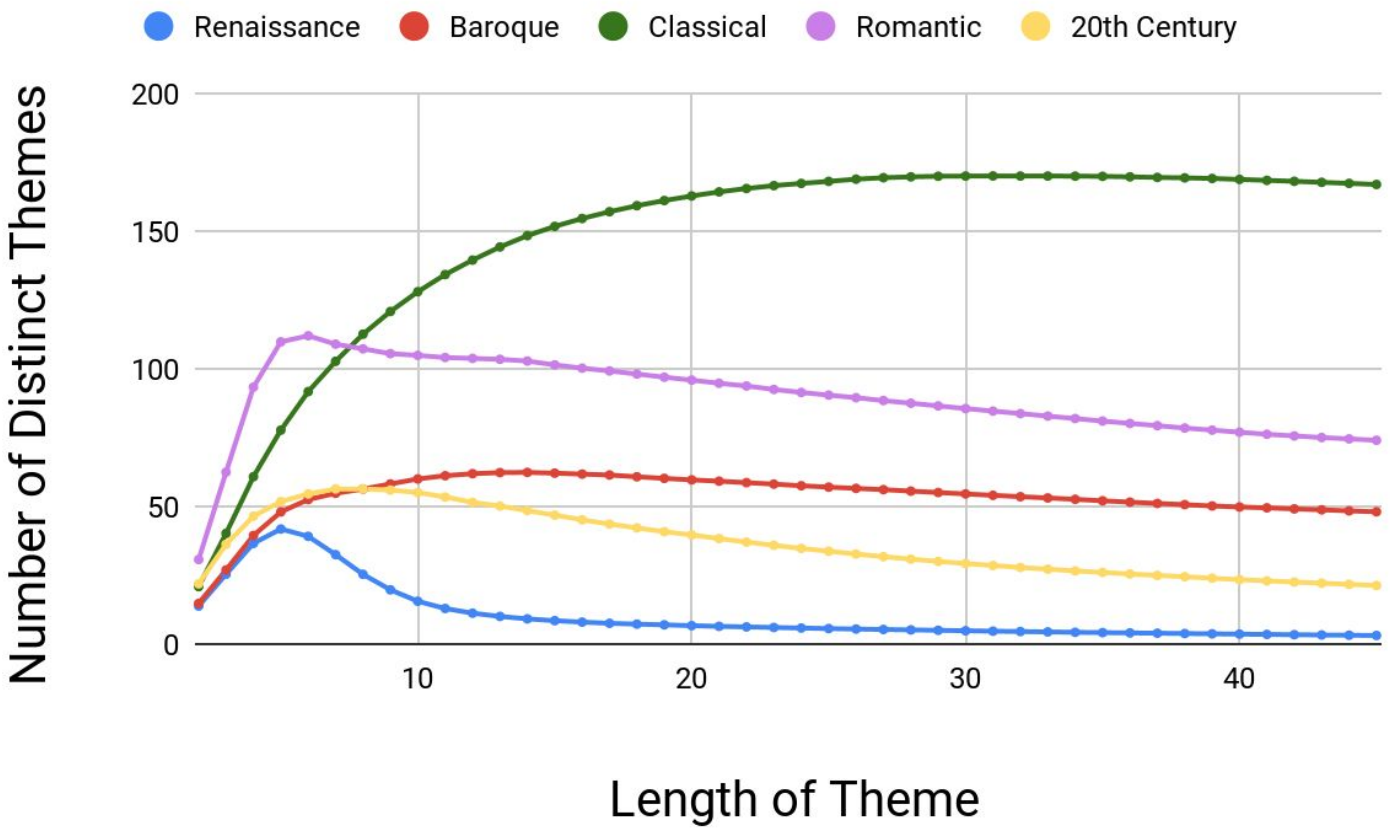
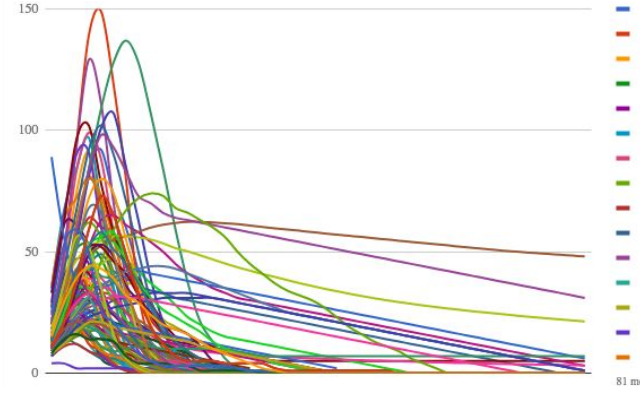


Fig. 1a: A graph showing the relationship between the number of themes and the length of themes. As we can see, each time period clearly has a unique curve. For example, Renaissance doesn't increase for long before it rapidly drops to near zero for many of the longer themes. Classical exhibits a very pronounced upward curve, while Romantic increases sharply before turning around and gradually dropping at around 6 notes long.

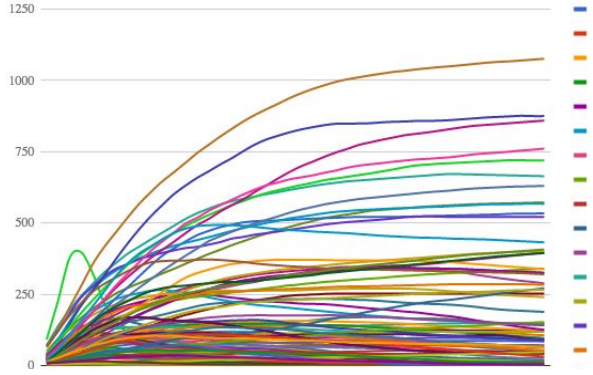
Rhythmic Themes (Cont)

In order to prove that these average curves actually represent the population of songs, we made graphs that showed all the songs in the time period. If you see the time period curves from the previous page, you can see how the songs generally fit the time period.

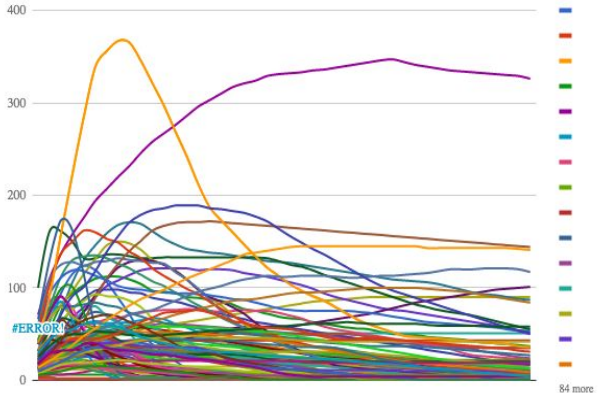
Themes over Length: Renaissance



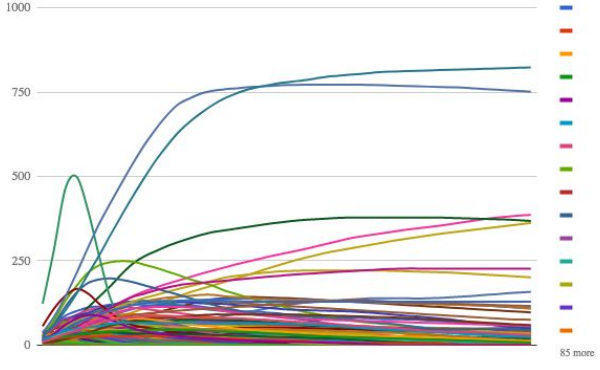
Themes over Length: Classical



Themes over Length: 20th Century



Themes over Length: Baroque



Themes over Length: Romantic

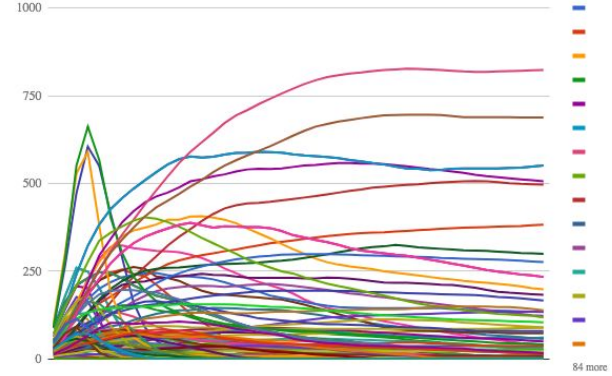


Fig. 1b: Five graphs each showing rhythmic themes over length for all the songs in a given time period. The Classical time period clearly all exhibit the Classical average curve. This could be why a lot of music in the classical era sounds similar. 20th Century, on the other hand, shows great diversity, probably due to the fact that music was becoming less conformist

Rhythmic Themes (Cont.)

Another aspect of themes that we analyzed for was how many times the theme was repeated. We could then graph the most amount of times a theme was repeated over the length of the theme.

As can be seen by the graph below, a distinct pattern emerges: several time periods, such as 20th Century and Romantic, exhibit a strange stepping pattern.

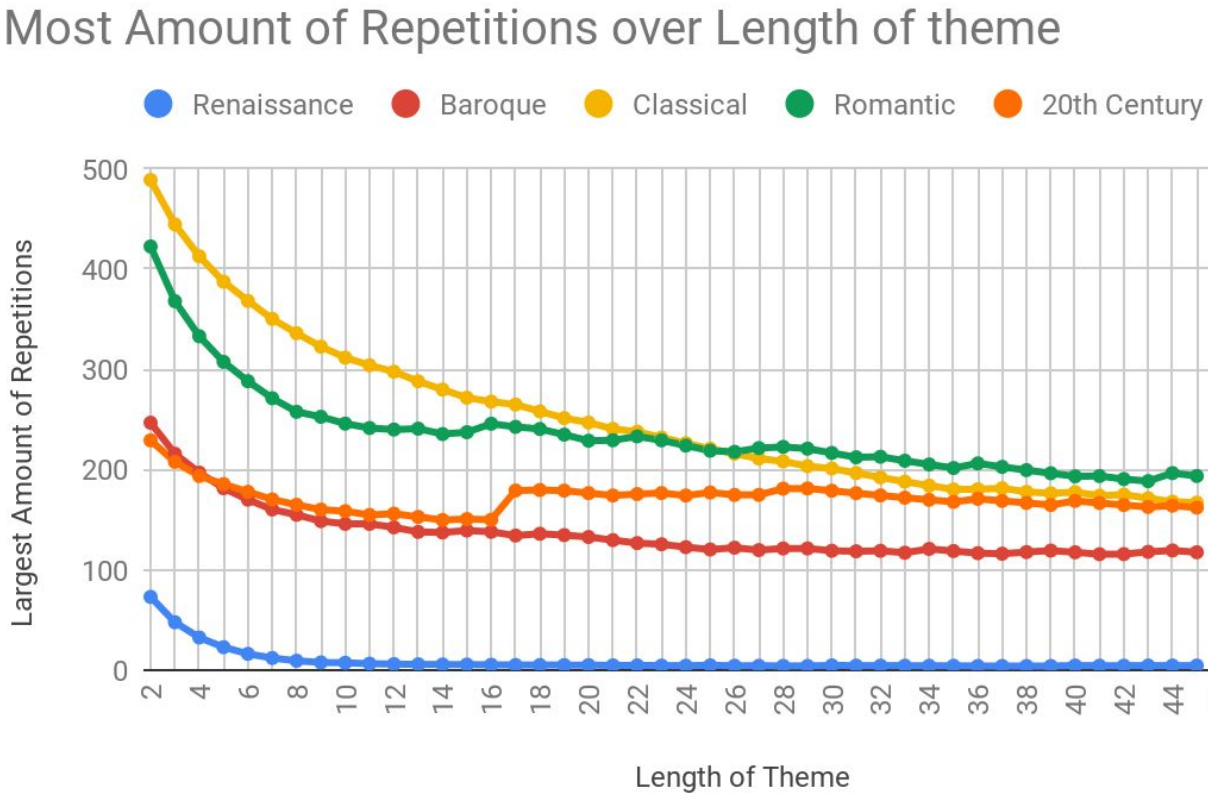
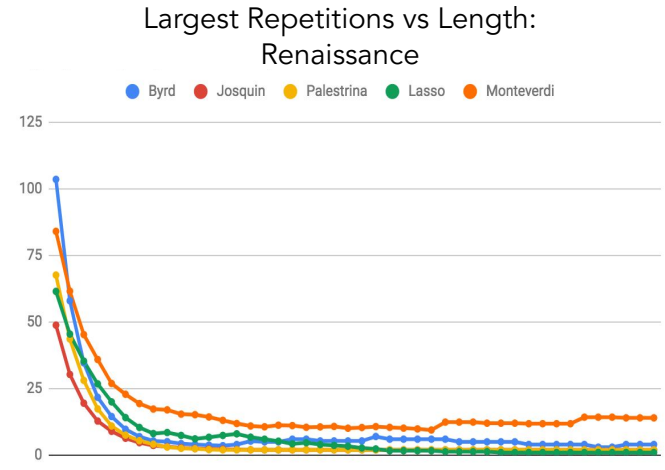
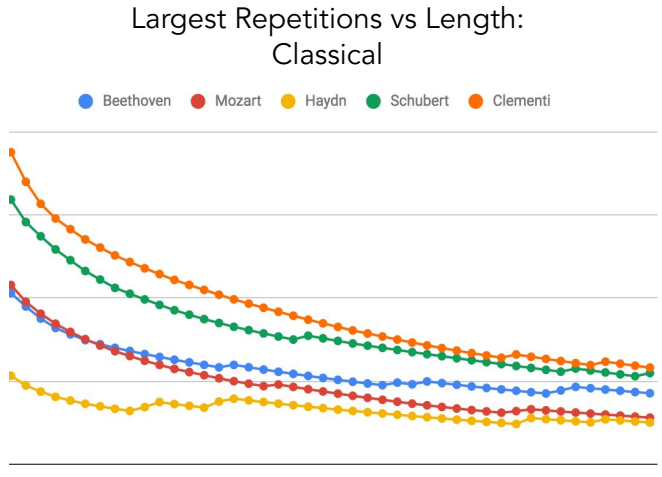


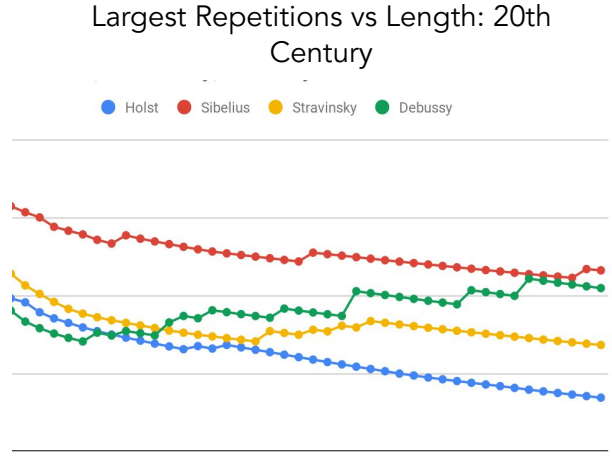
Fig. 2a: A graph of the largest amount of repetitions over the length of the theme for each of the different time periods. A strange stepping pattern can be seen in all of these, appearing like a staircase. A value will suddenly increase, and then linearly fall until the next increase.This is especially apparent in 20th Century and Romantic. This was an extremely surprising finding, as there is no possible explanation that we thought of that would explain this phenomenon.

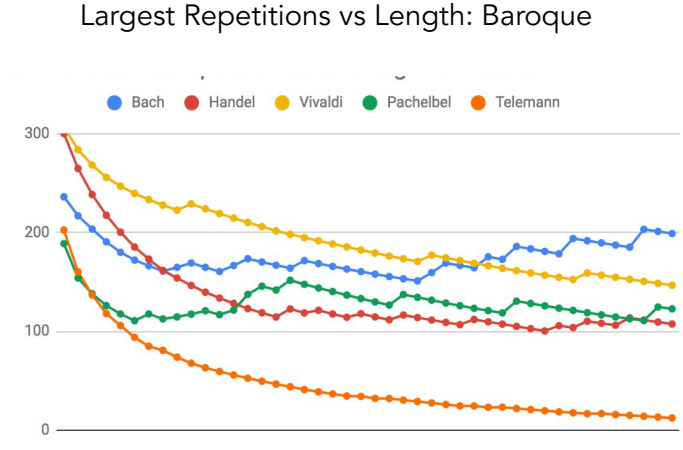
Rhythmic Themes (Cont.)

We then tested this strange stepping phenomenon on the individual composers. We graphed each of the composers in a time period together (with five graphs total). We found the pattern to be even more pronounced:









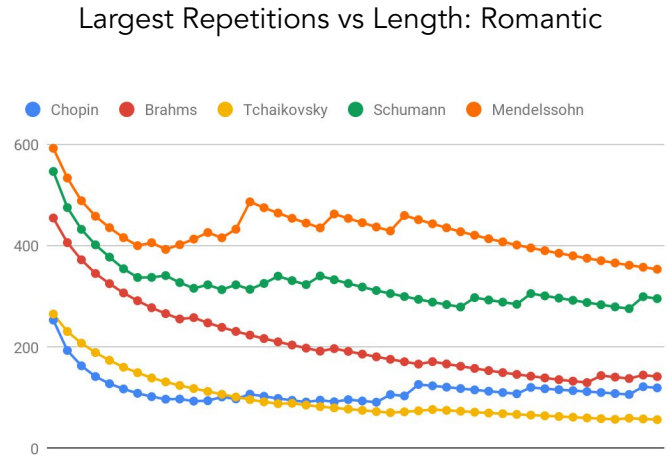


Fig. 2b: A display of the composers from each time period. As can be seen, there is an incredible 'stepping' phenomenon very visible in Romantic, Baroque, and 20th Century music. It is less pronounced in Classical, and hardly visible in Renaissance. This could be due to the Renaissance's early time.

Rhythmic Themes (Cont.)

The final curve we decided to look at was that of the total number of repetitions of all themes of a given length. This ended up giving us very smooth distributions, as pictured below. While all of them descend, they descend at different rates, the most notably of which is Romantic, which starts with the highest number of repetitions but drops below Classical beyond 6 note themes. Baroque and 20th Century start off differently to begin with but then consistently narrow as the length of the themes grow.

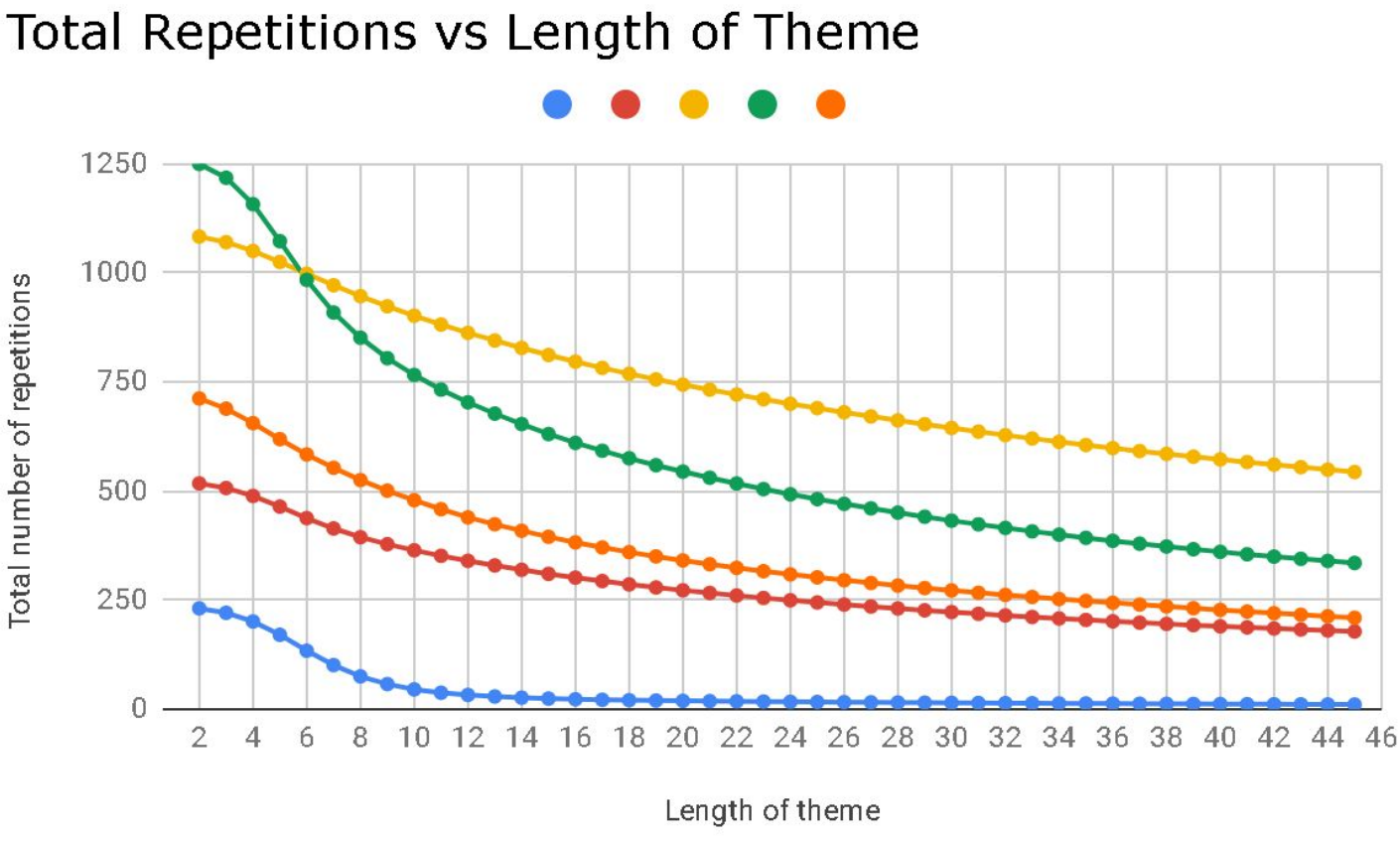
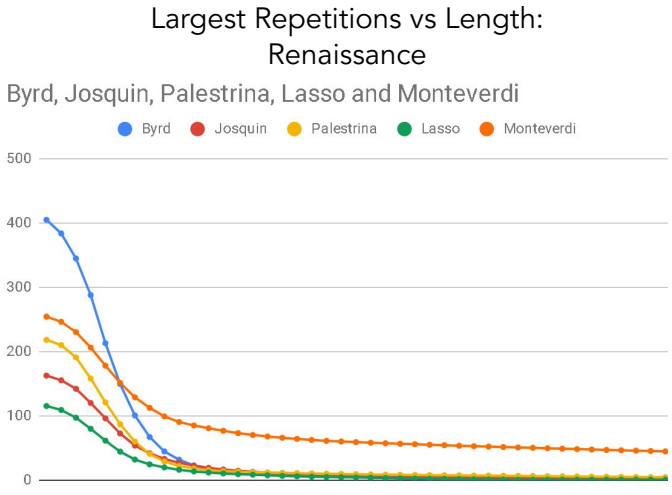
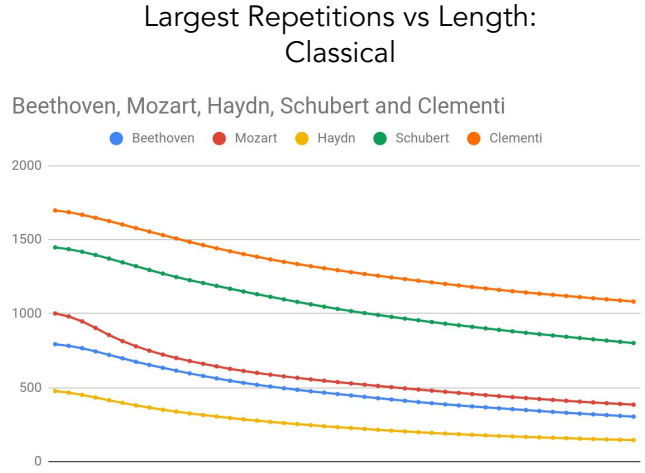


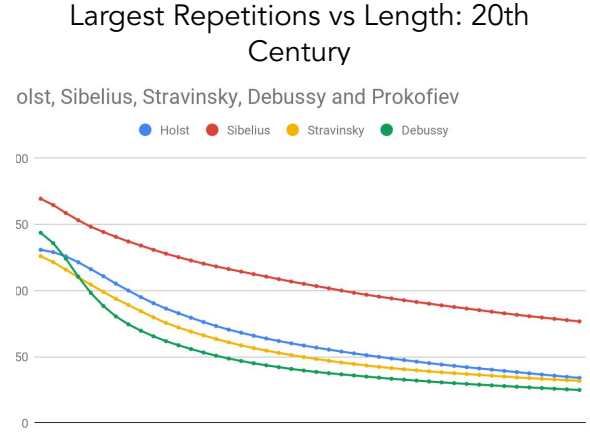
Fig. 3: A line graph shows the total repetitions over the length of theme plotted for the 5 time periods (standard coloration). There are some similarities between this graph of total repetitions vs. the most amount of repetitions of one theme graph, pictured two previous pages ago. For example, Renaissance similarly drops to near nothing at and beyond 10-12 note themes (and 6 note themes for most amount). Additionally, Romantic also drops below Classical in most used themes, though much later than in total repetitions.

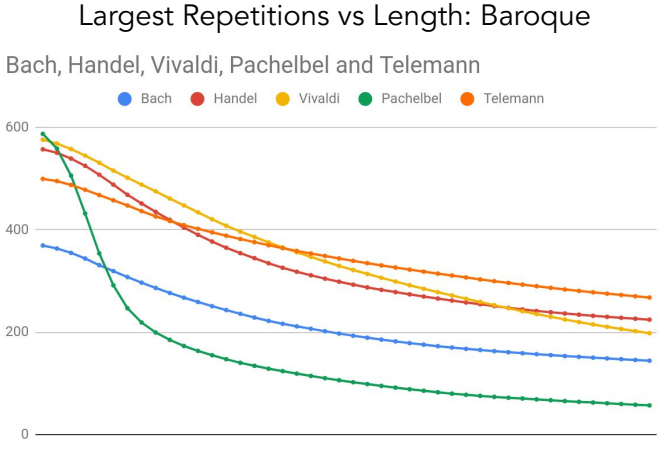
Rhythmic Themes (Cont.)

We also measured the number of repetitions of the most repeated theme. We found that the this also creates very unique curves that fit well with equations.









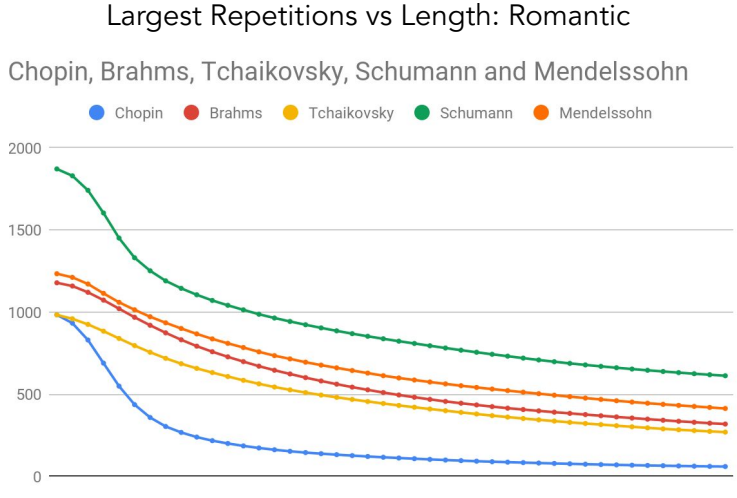


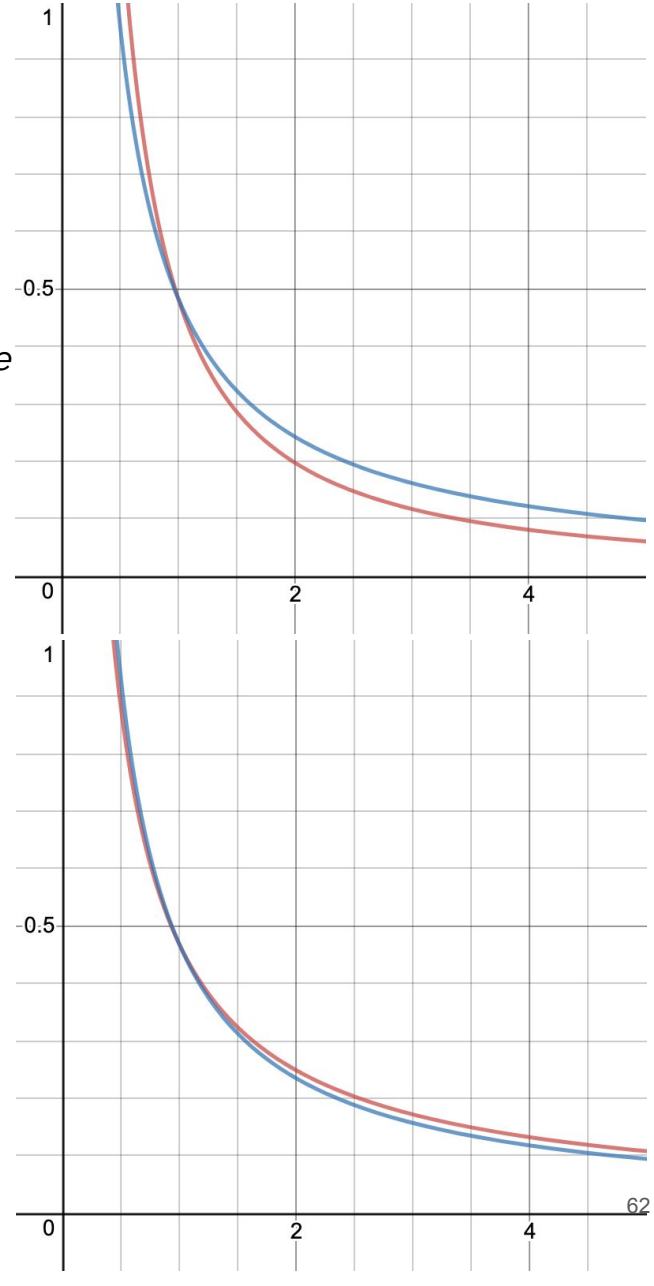
Fig. 4: Graphs of each of the time periods show distinctive and unusual trends. Unsurprisingly, Renaissance starts low and quickly drops (though Monteverdi does consistently have a much higher number of themes beginning at 7 note themes). In Baroque, Pachelbel begins with the highest but also rapidly drops, unlike any other Baroque Composer.

Most Used Note Values

The graph on the right shows the Romantic time period in red and Zipf's Law in blue. Romantic has an *a* value of 1.298, while Zipf's Law is always 1. However, this is measured in notes, without accounting for the length of each one. When you *do* factor this in, the distribution stays the same. In fact, the values themselves also stay the same to the point that the null hypothesis can be accepted because of a two-tailed *p* value of 0.34. Measuring in time, the graph looks much more similar to Zipf's Law.

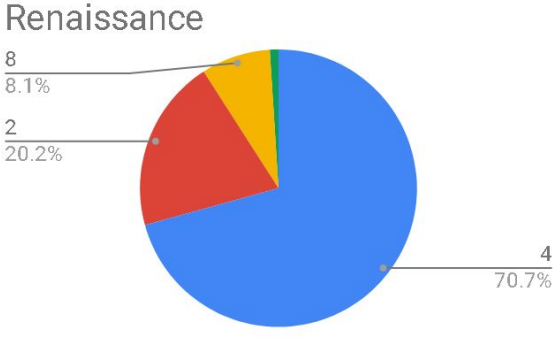
Fig. 1a (Right): In red- The curve measuring occurrences over usage rank for the most used note values during the Romantic time period. In blue- Zipf's Law.

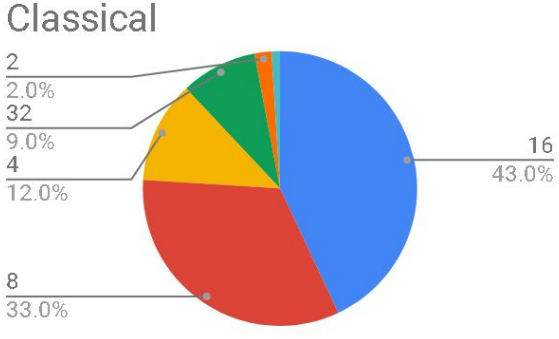
Fig. 1b (Right): In red- The exact same measurement as in fig. 1a, only the length of the note value is factored into this data. In blue- Zipf's Law.

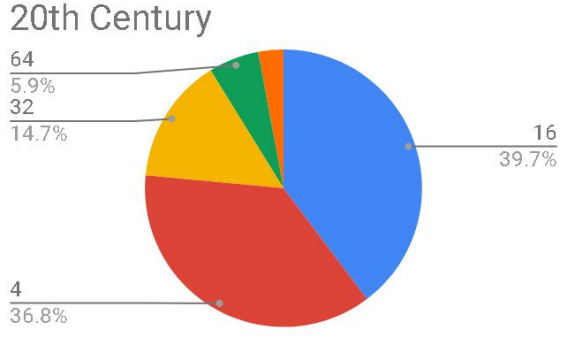


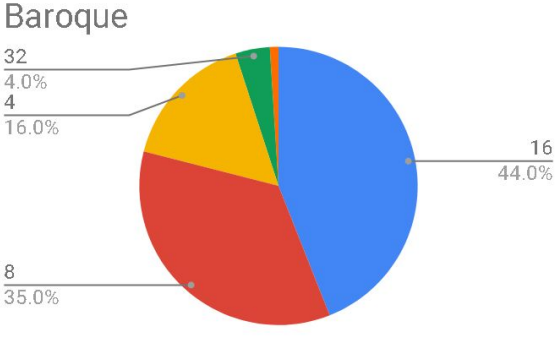
Most Used Note Values (Cont)

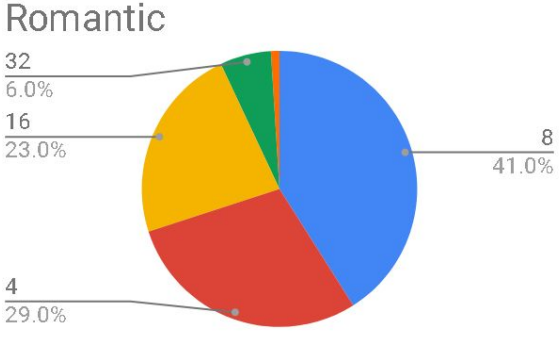
We also created pie charts for all 5 time periods and classical music to depict the distribution of different note values solely based on occurrences, or the average percent of the notes in a song in a specific note value. Each time period has varying distributions. Not surprisingly, Renaissance's top 3 values dominate 98% of the average song, continuing the trend of steep distributions for Renaissance, while the top 3 values take up just 87% in Classical. Overall, classical music's top 3 values take up 87.9%, slightly more than in Classical. Several values show up quite often in the top 3, especially sixteenth notes (denoted by 16 here), which is the top note in three time periods.

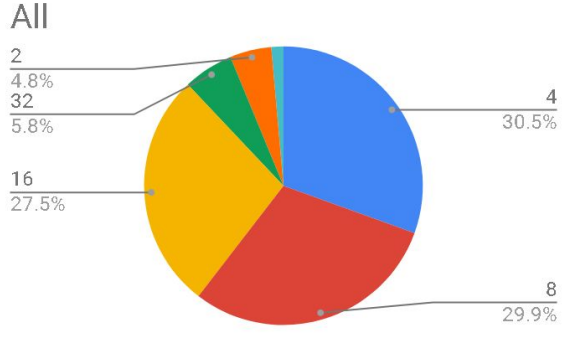












Most Used Pitches

There are actually two curves present in the graph to the right. The blue one is Zipf's Law and red is most used pitches for the Renaissance time period (although you pretty much can't see it. The equations are identical except for a difference of 0.018 in their a values. We also looked for Pareto distribution, and found that there were no cases of the exact 80/20 rule, we found a 80/42 rule when including all of our data. This pie chart of classical music shows that the top 42% of pitches make up exactly 80% of of the songs' most used pitch (there are 14 pitches, and the top 6 are pictured here). We used scale degree, so the 5, 4, and 1 pictured in the pie chart are the dominant, subdominant, and tonic respectively, very common notes.

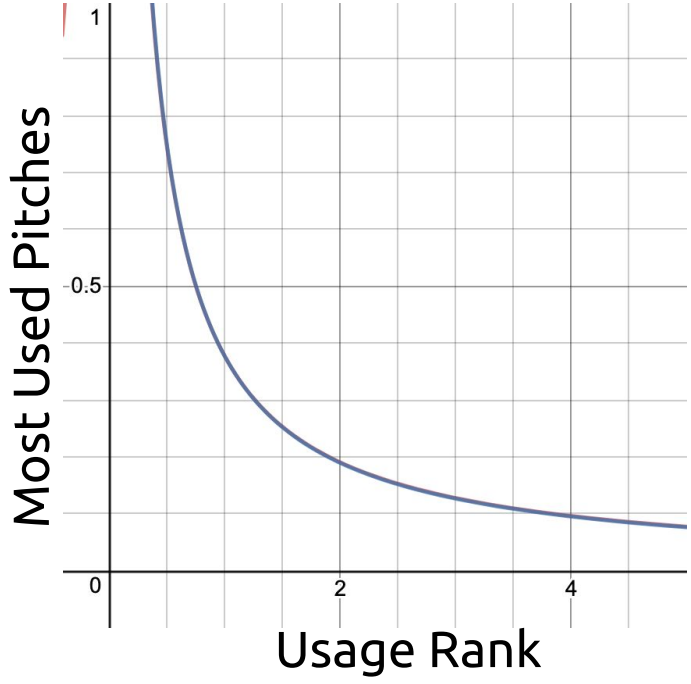


Fig. 1: The visible blue curve in this graph is Zipf's Law, but the curve for Renaissance is also present, but hidden by the other curve.

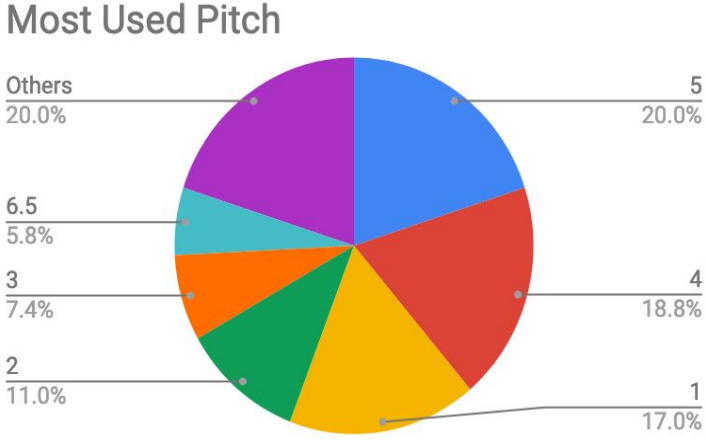


Fig. 2: A pie chart showing the Pareto distribution in our data.

Most Used Pitches (Cont.)

Having found the first through seventh most used pitches of each song, it was natural for us to analyze these values in comparison to one another.

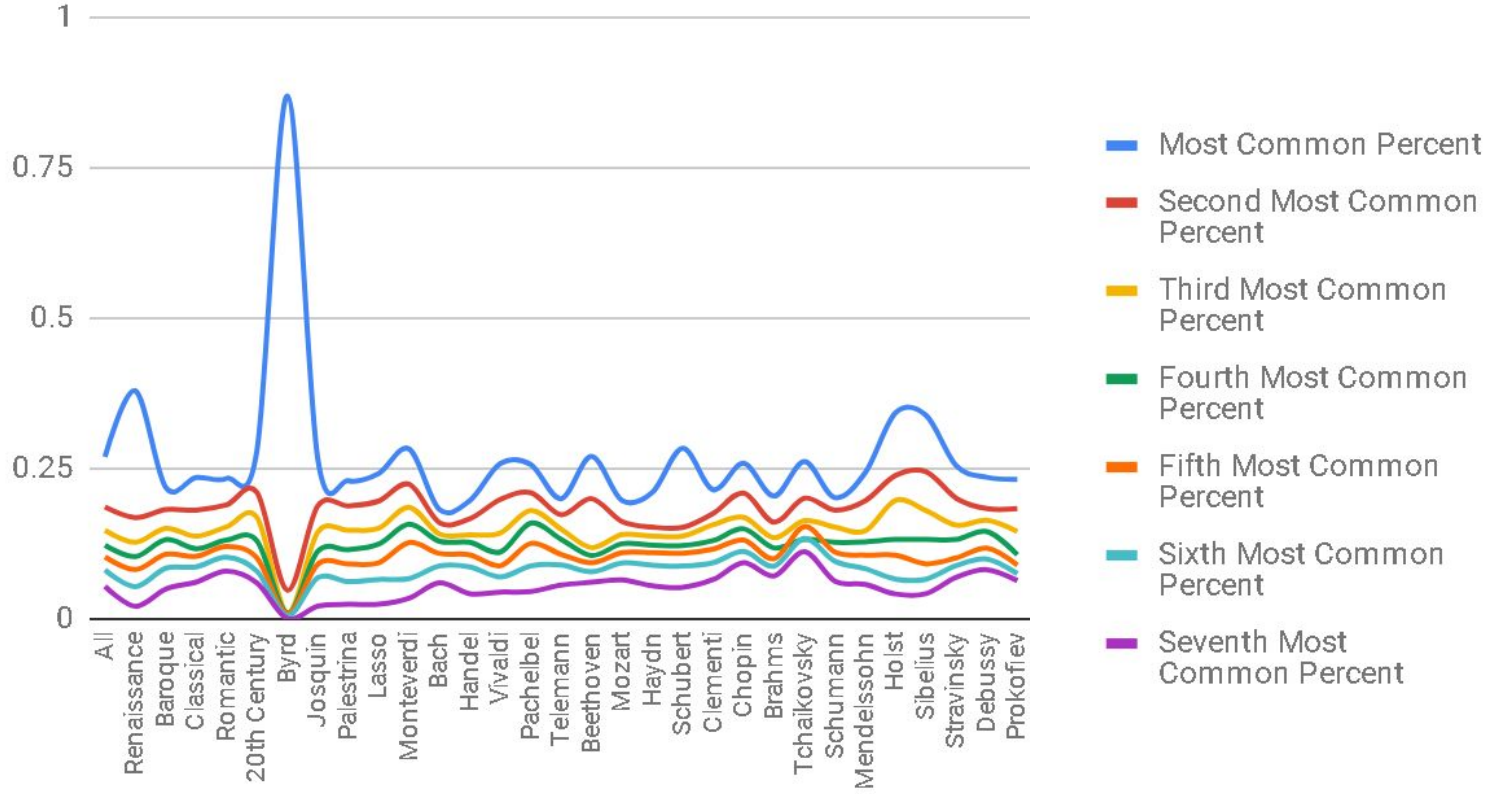


Fig. 3: A line chart showing the average percent of song in e itch for each composer. Byrd is an extremely unusual composer, with 67% of his songs in the same pitch on average, as compared to 27.9% for all of classical music. In fact, removing Byrd from the averages drops the average percent of song in the most common pitch to around 25%, as compared to almost 28% with his music included.

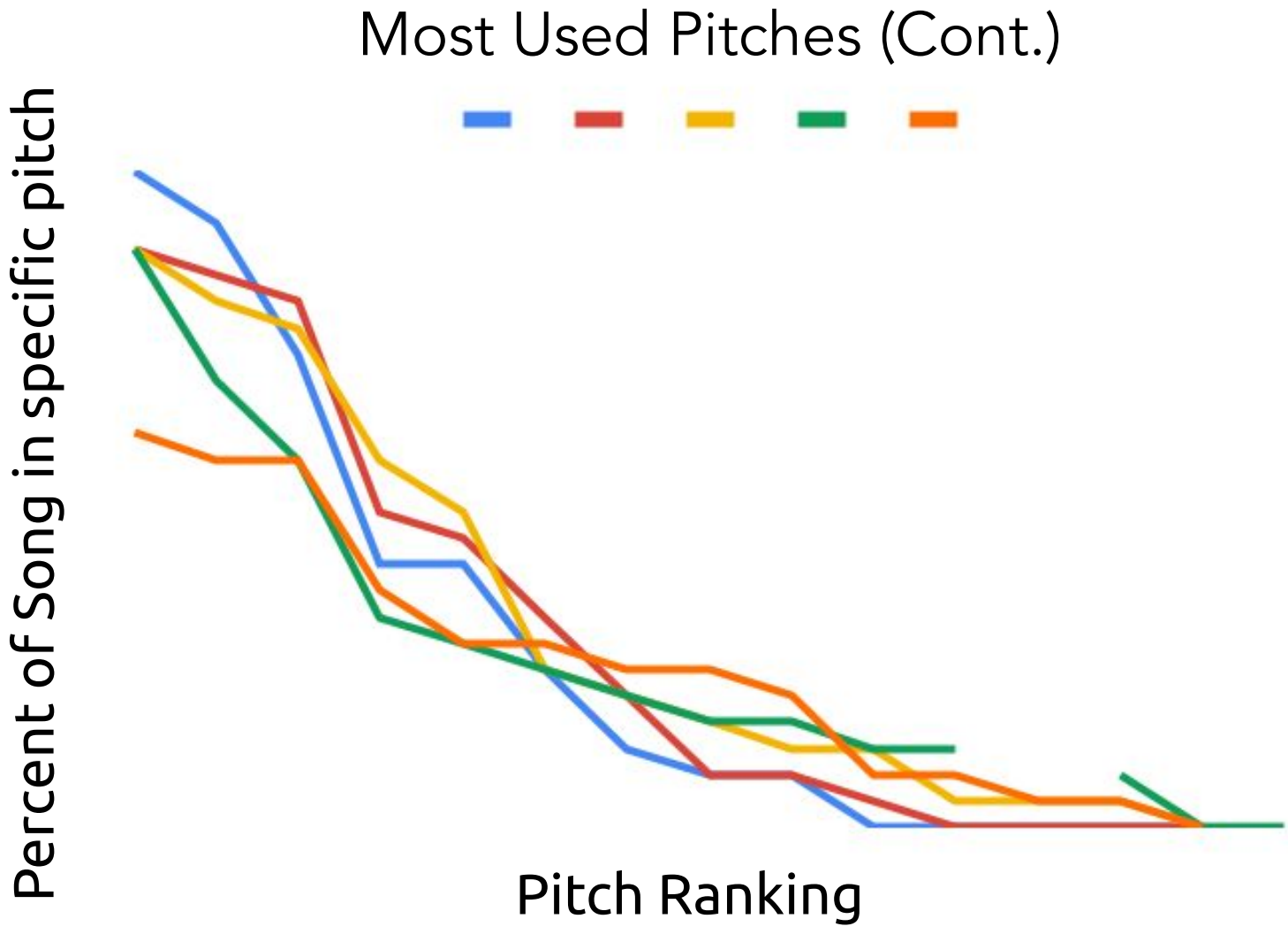


Fig. 3: A line chart showing the overall drop of the average percent of song in a specific pitch, by time period. Renaissance starts with 25% of their songs having the same pitch, more than in any other time period. The middle 3 time periods all start at the exact same pitch, however Romantic starts dipping faster than the other two, before slowing down later (it is also the victim of a weird glitch that created a break in the line).

Analysis of 80/20 rule in Most Used Pitches

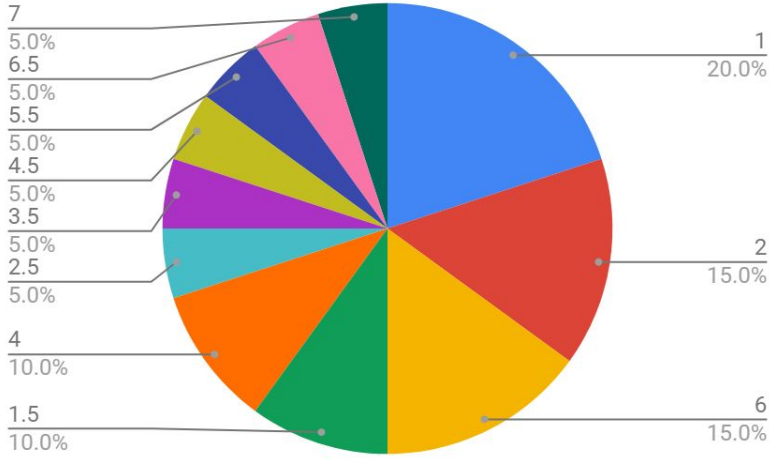
We analyzed for 80/20 rule in the most common pitches by composer, time period, and classical music in general. Taking all of the songs and finding their most common pitch allowed us to find which pitches were most common as the most common pitch. From here, since there are 14 pitches, we took the top 3 pitches (3/14~21%) and found about how many songs listed those three pitches as their most common pitch. We then found whether those songs made up close to 80% of the songs (anything from 75% to 85% was accepted). We found that 7 composers out of the 25 we analyzed followed the 80/20 rule, 2 in Renaissance, one in Baroque, three in Classical, none in Romantic, and one in 20th Century.

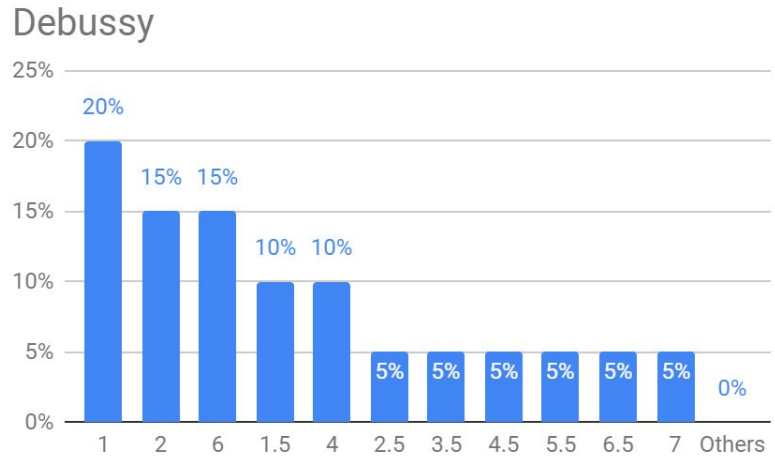
Fig. 1: A table shows the composers that followed the 80/20 rule and their three most common pitches. The number of songs that used one of three pitches was close to 80%, thus qualifying that composer as following the 80/20 rule. (The table uses scale degree).

	Most Common	Second Most	Third Most
Byrd	4	5	2
Josquin	1	4	5
Pachelbel	4	2	5
Beethoven	2	5	4
Mozart	5	1	2
Clementi	1	5	6
Prokofiev	2	1	5

Analysis of 80/20 rule in Most Used Pitches (Cont.)

However, the large majority of composers didn't follow the 80/20 rule. Of these composers, of which there were 18, all but one of them undershot the 80/20 rule. In other words, taking 20 percent of the pitches took up less than 80% of the songs, and it took more than 20 percent of the pitches to take up 80% of the songs. The first is denoted by XY/20, where the XY is some number representing what percent of the songs were taken by those pitches, and the latter is denoted by 80/XY where the XY is the percent of the pitches needed to get 80% of the songs (or close to it). If a composer followed 80/20 rule, obviously the XY/20 and 80/XY were the same, but if not that could be different, e.g. 65/20 and 80/42.

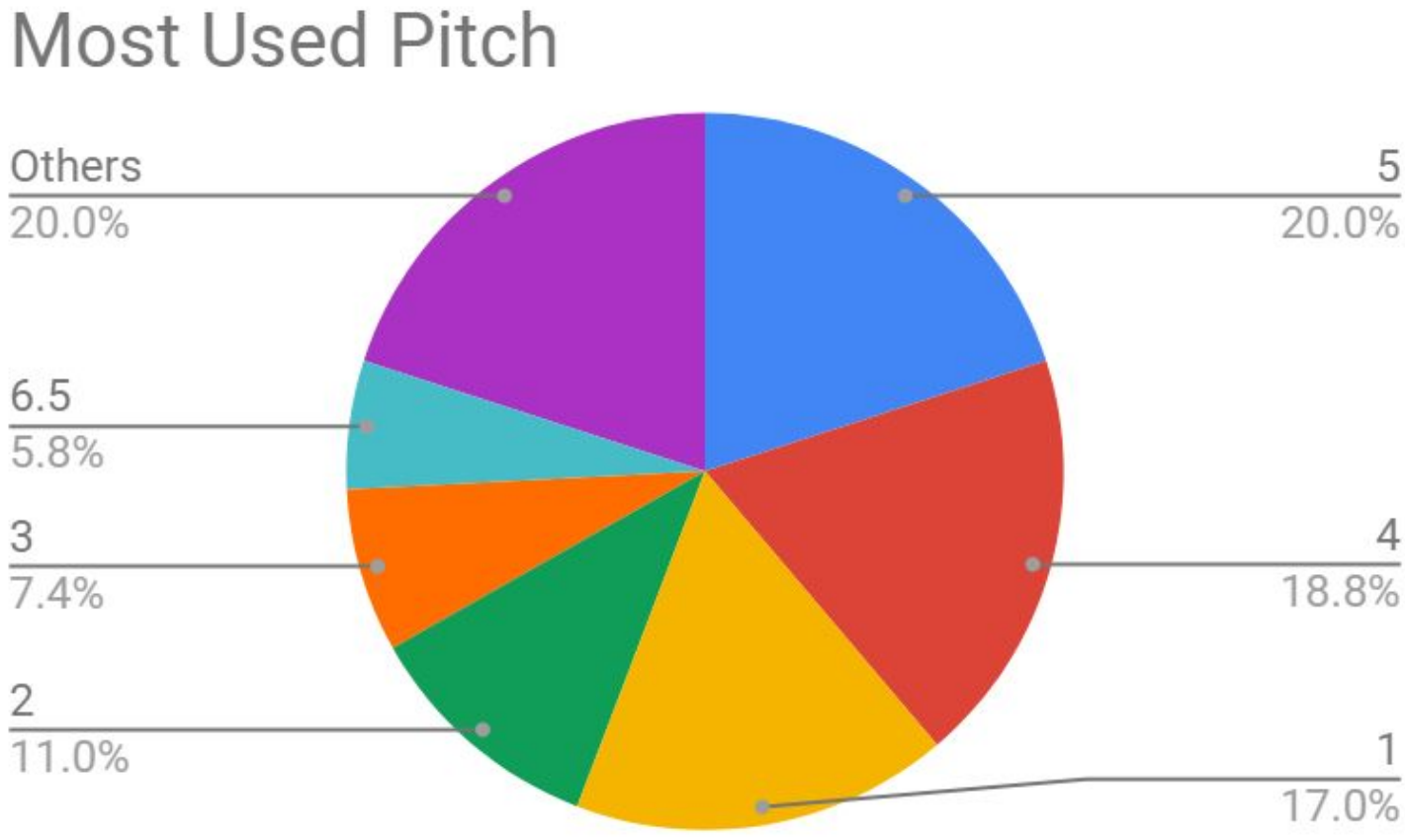




Figs 2a and 2b: Two different types of graphs, one a pie chart, the other a column chart, paint a different picture of Debussy's most common pitch in his songs. In the pie chart, you can see Debussy follows 50/20 and 80/50. The bar chart shows how much flatter his distribution of most common pitches is (at least compared to some composers.)

Analysis of 80/20 rule in Most Used Pitches (Cont.)

This pie chart shows the percent of songs with a specific pitch as their most used pitch, for all of classical music. As you can see, the dominant (5), subdominant (4), and tonic (1) take up a large amount of the songs, between 17-20%, before dropping quickly to the supertonic and mediant (2 and 3 respectively). The dominance of 5, 4, and 1 is not entirely surprising as they make up the infamous chord-progression I-IV-V (Roman numerals for 1-4-5).



Analysis of 80/20 rule in Most Used Pitches (Cont.)

From here, we converted the fractions into decimals to graph them.

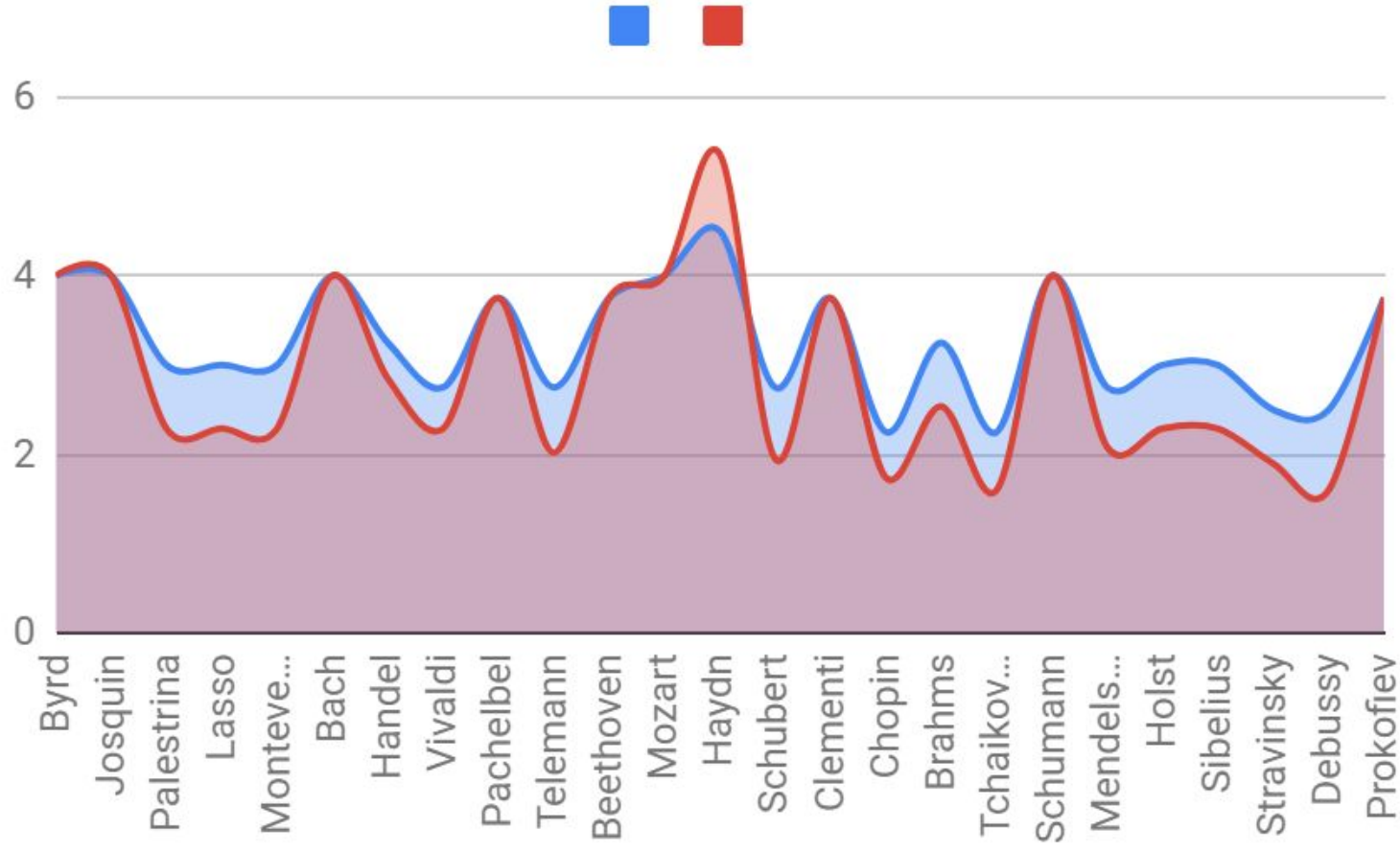


Fig 3: From here, we graphed both XY/20 and 80/XY. The red is 80/XY, and the blue is XY/20. As can be seen, the composers that follow the 80/20 rule are the ones where the blue and red areas meet, and the composers that don't all have a lower 80/XY, with the exception of Handel, who not only overshoots the 80/20 rule for both XY/20 and 80/XY but also is the only composer where 80/XY is higher than XY/20.,

Analysis of 80/20 rule in Most Used Pitches (Cont.)

We also turned this into a scatter plot, with the x axis being XY/20 and 80/XY being the y axis. After noticing the curve, we turned it into a log-log graph and found some interesting results.

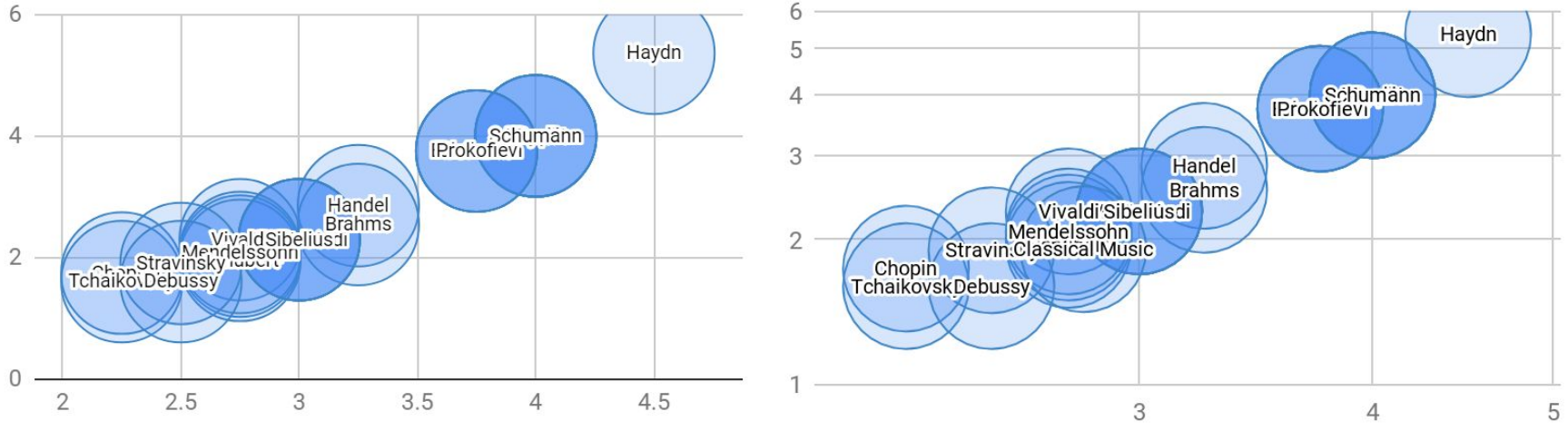


Fig 4: There are three segments of importance in both graphs. Haydn is alone in being the only composer that overshoots the 80/20 rule, while the composers that follow the 80/20 rule, represented by Prokofiev and Schumann (though there are actually more composers than it looks that follow the 80/20 rule, but they occupy the same spot so can't all be shown. As to the composers that don't follow the 80/20 rule, as well as classical music, there is a lot of diversity and overlapping circles as many composers occupy an individual spot on the scatter plot.

Average Pitch

The average pitch throughout all the time periods was surprisingly consistent, almost always within the supertonic and mediant (the second and third notes of a scale). This is all relative to the tonic (1st note note in scale), so in these measurements, a low “A” and a high “A” mean the same thing. This could be explained by frequent usage of the tonic and dominant (5th note in scale) in music, because the averages tend to fall about

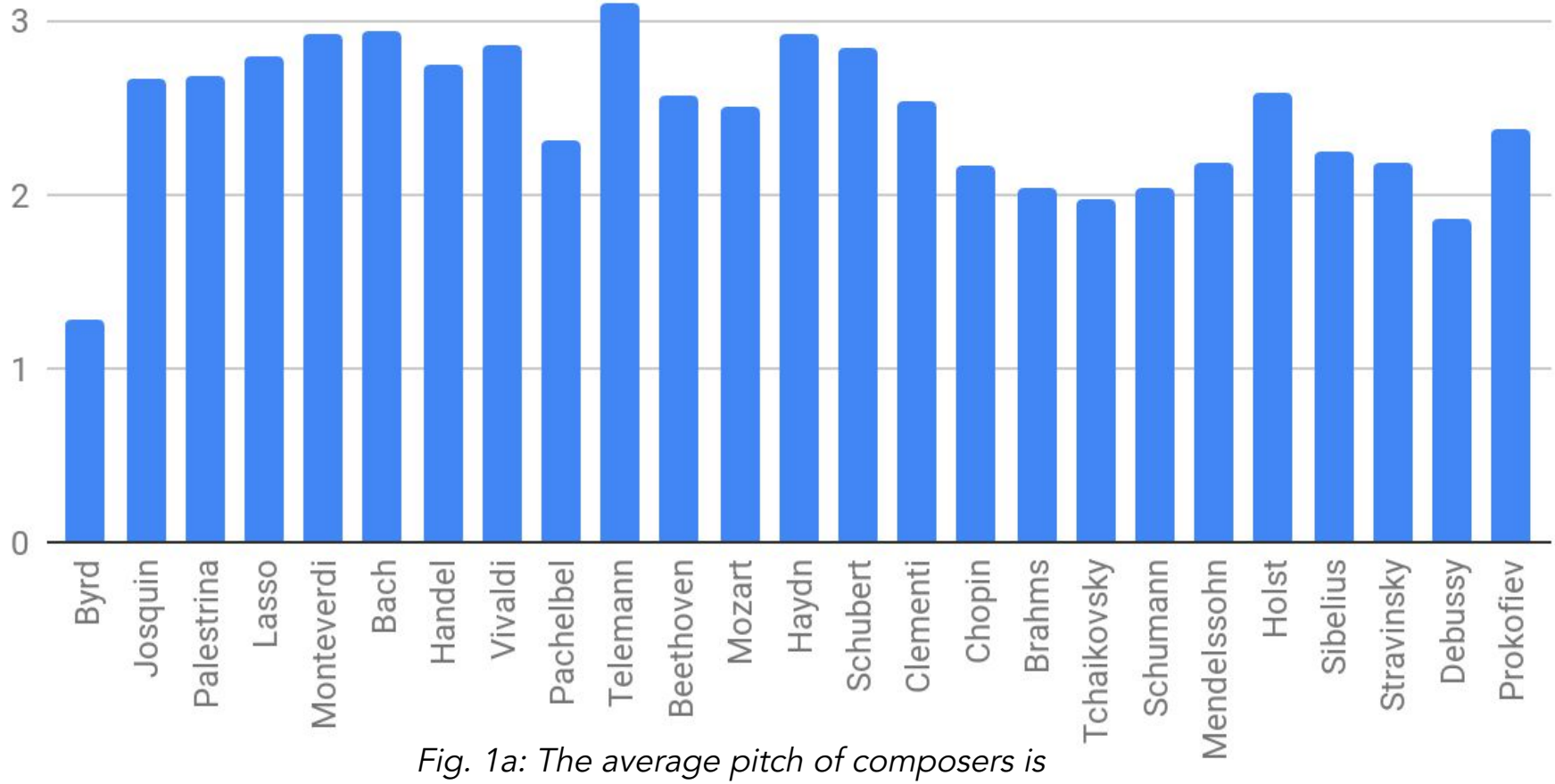


Fig. 1a: The average pitch of composers is relatively uniform, with an exception of Byrd.

Average Pitch (Cont.)

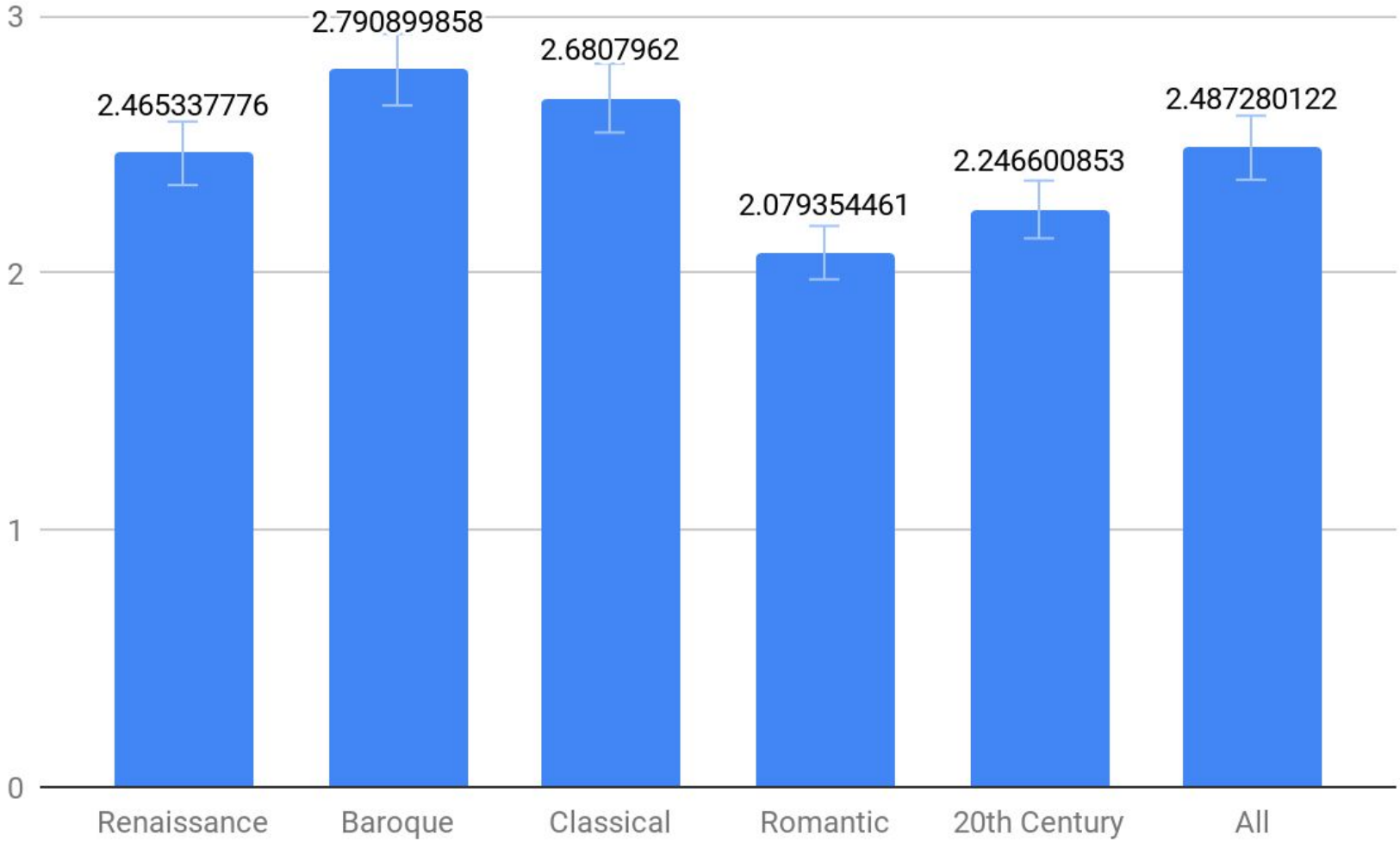


Fig. 1b: The same measurement done with time period shows a similar evenness. Romantic has the smallest average pitch, while Classical has the highest average pitch. This may reflect a larger use of the subdominant (4th note in scale) and dominant, as well as higher pitches, compared to Romantic. Renaissance is just under the average pitch, with an average pitch for Renaissance of around 2.465 compared to an overall average pitch of 2.487 for all of classical music.

Average Pitch (Cont.)

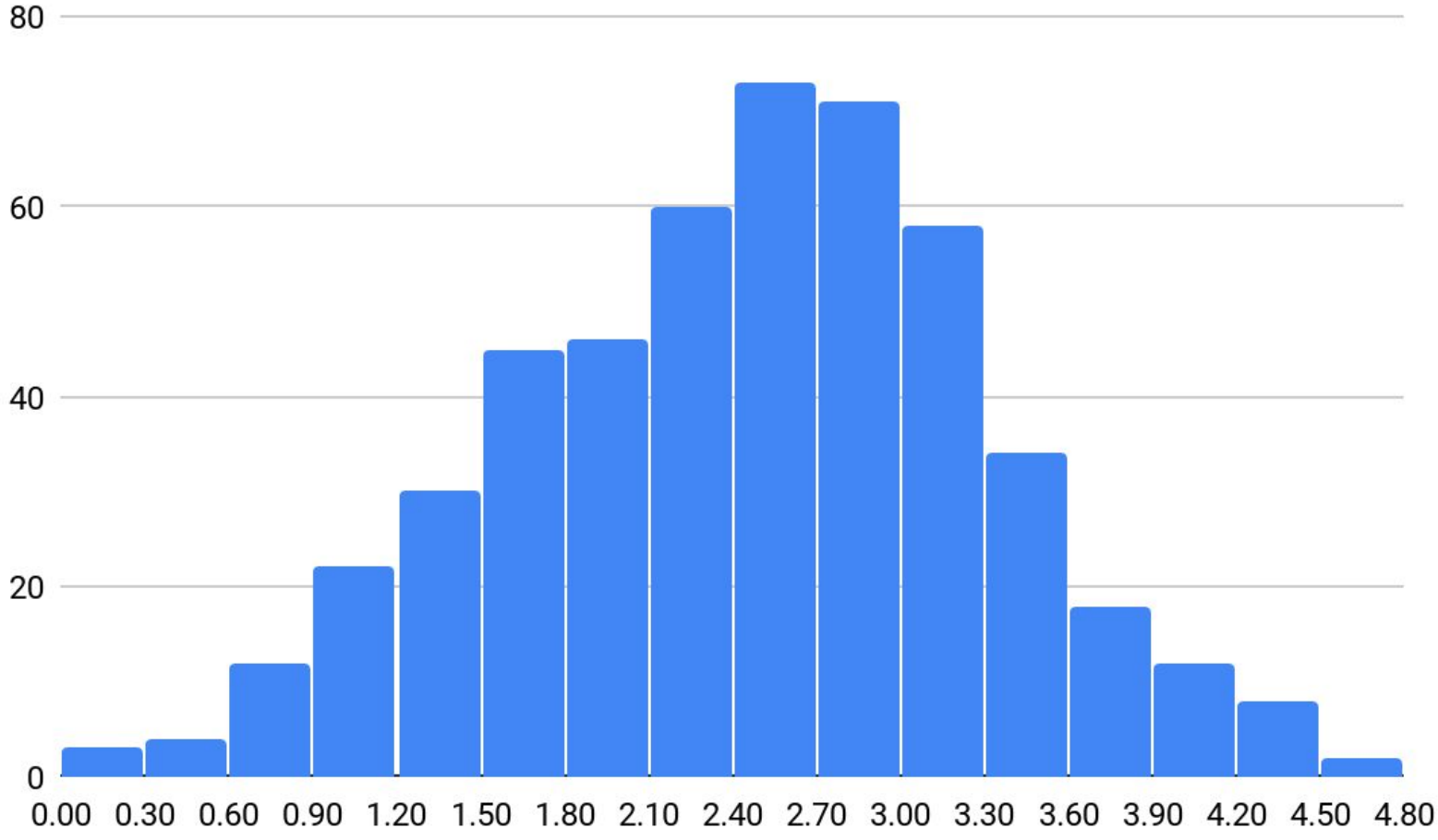
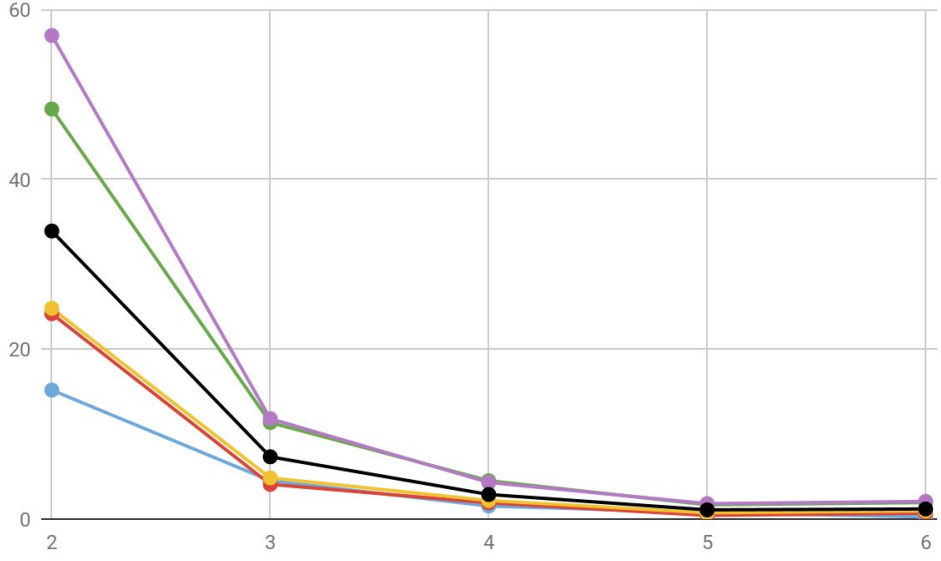


Fig. 2: A histogram of the average pitch of each song shows a near perfect bell curve and normal distribution. The data had an extremely low skewness (below ½ is considered very symmetrical) of 0.4500512626. Additionally, analyzing for kurtosis also gives an acceptable value, namely -0.205, with a standard error of 0.11 for the skewness and 0.22 for the kurtosis. Additionally, running an Anderson-Darling normality test gives a confidence of 3.38%, within our 95% confidence level.

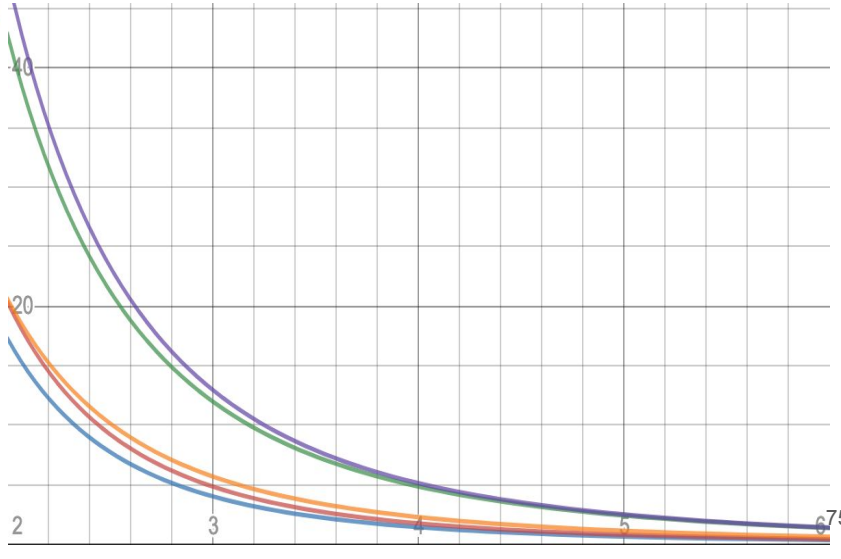
Sequences of Repeated Pitches

We also analyzed for the number of sequences of repeated pitches. This allowed us to then graph the number of sequences over the length of the sequence for each of the composers. Similar to scales, we got a very distinct power law curve for each time period.



Renaissance	$y = 210.461529x^{-3.604959634}$
Baroque	$y = 237.630139x^{-3.545606647}$
Classical	$y = 379.928806x^{-3.14668177}$
Romantic	$y = 447.125714x^{-3.223811479}$
20th Century	$y = 187.544491x^{-3.183106488}$

Fig. 1a (left): A graph showing the number of themes (y) in relation to the length of the theme (x). While we also found 7 note themes, they were disregarded as that would skew our results if more than one length was counted as just one length. Each curve represents a time period. Fig. 1b (right): The same graph as fig. 1a, except represented as power law with specific equations. The x axis is 2 different compared figure 1a as 2 note themes are 1 on the x axis and so on.



Sequences of Repeated Pitches (Cont.)

We also noticed that when graphing the standard deviation and the mean, we get parallel curves. Though this is generally to be expected, we were intrigued that the more sequences you have, the more diverse they get.

Fig 2 (Below): We also found correlation coefficients between the amount of sequences of two different lengths. This chart displays the correlations between the two variables listed. As they get larger, correlation goes down.

Length 1	Length 2	Correlation
2	3	0.6511710766
3	4	0.4753448761
4	5	0.4830176237
5	6	0.4013248445

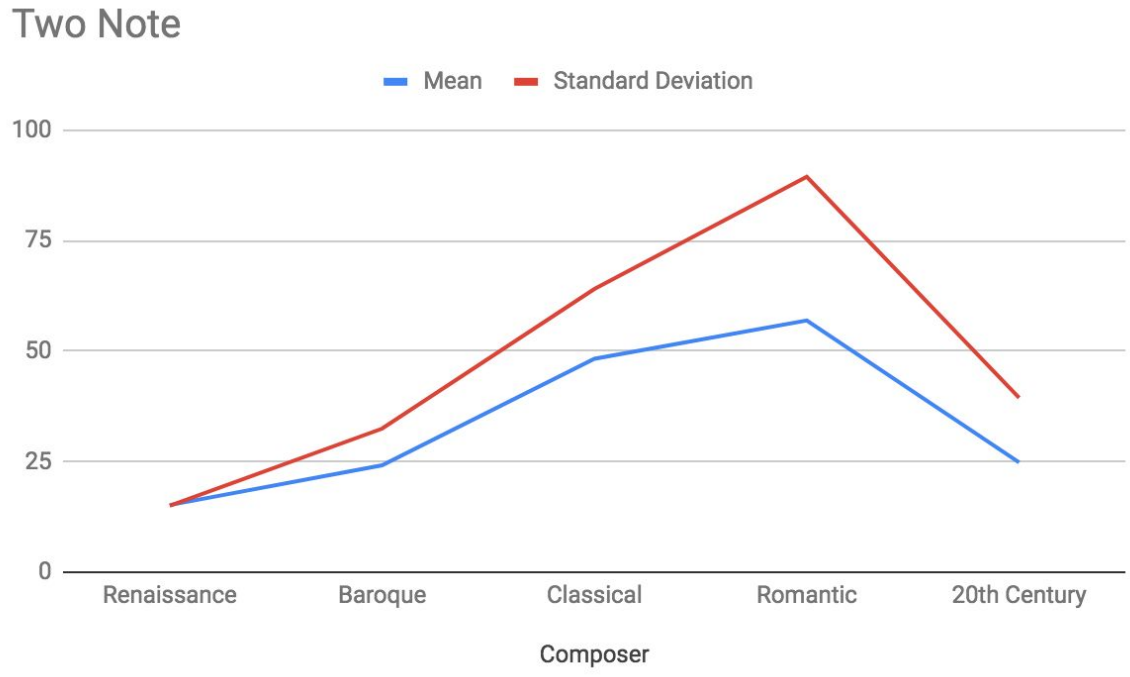
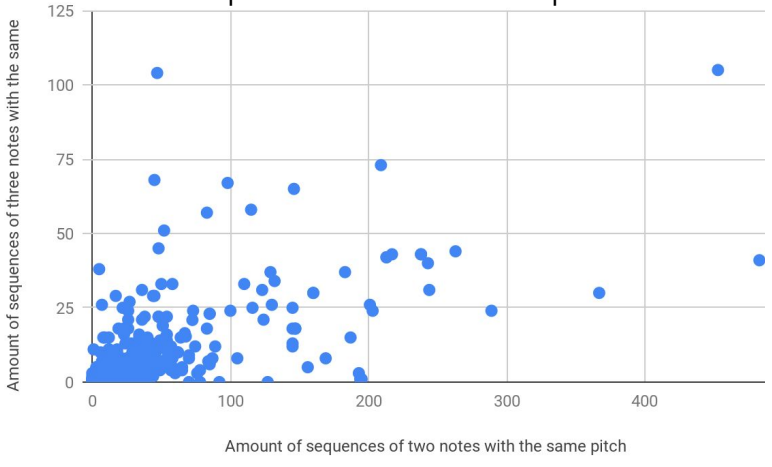


Fig 1 (Above): A graph of standard deviation and mean. We can see that they form parallel curves.

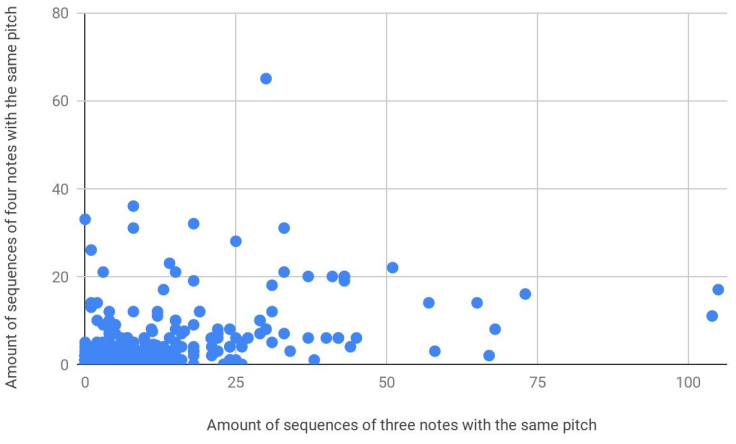
Sequences of Repeated Pitches (Cont)

We also made scatter plots comparing two different lengths of themes for the entire progression of the lengths, as shown below.

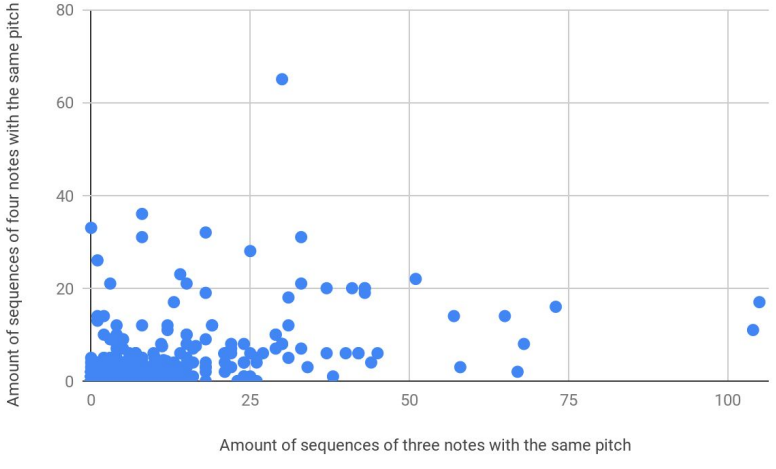
3 note sequences vs 2 note sequences



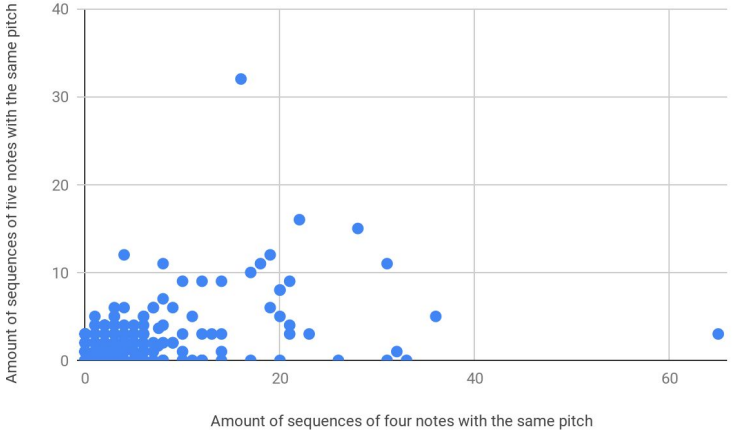
4 note sequences vs 3 note sequences



5 note sequences vs 4 note sequences



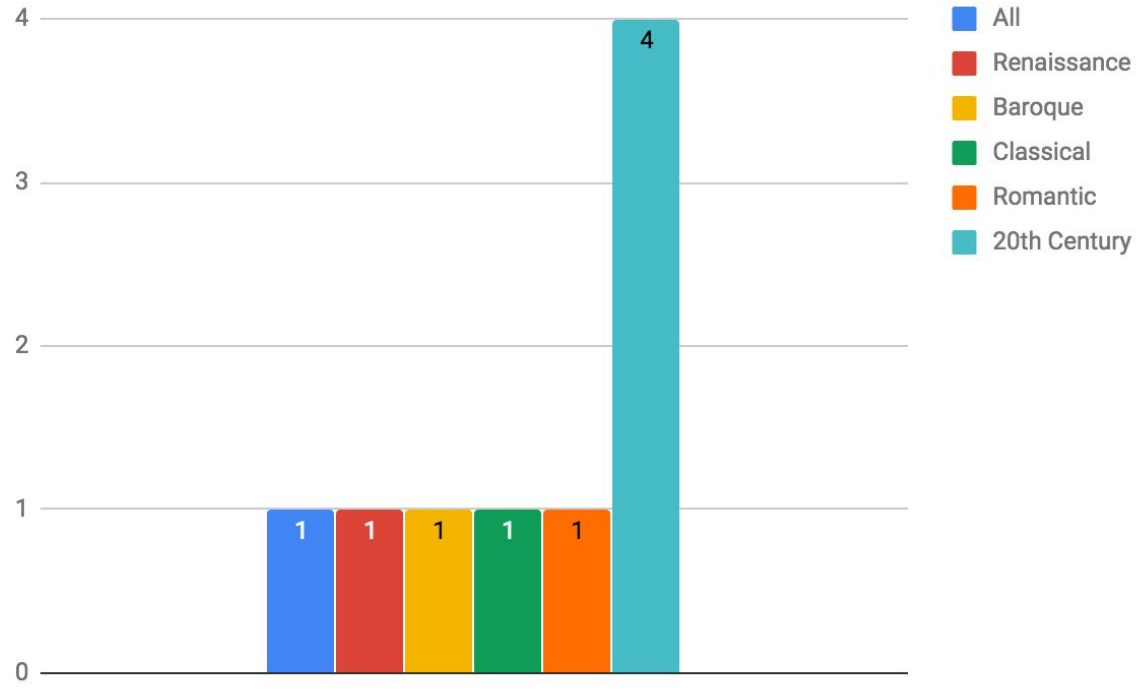
6 note sequences vs 5 note sequences



We can see from these plots that as the values get larger, the concentration of the data becomes flatter, as all the values move lower. This means that there are increasingly less n+1 length sequences than n length sequences. We can also see that the vast majority of the data is concentrated near the origin.

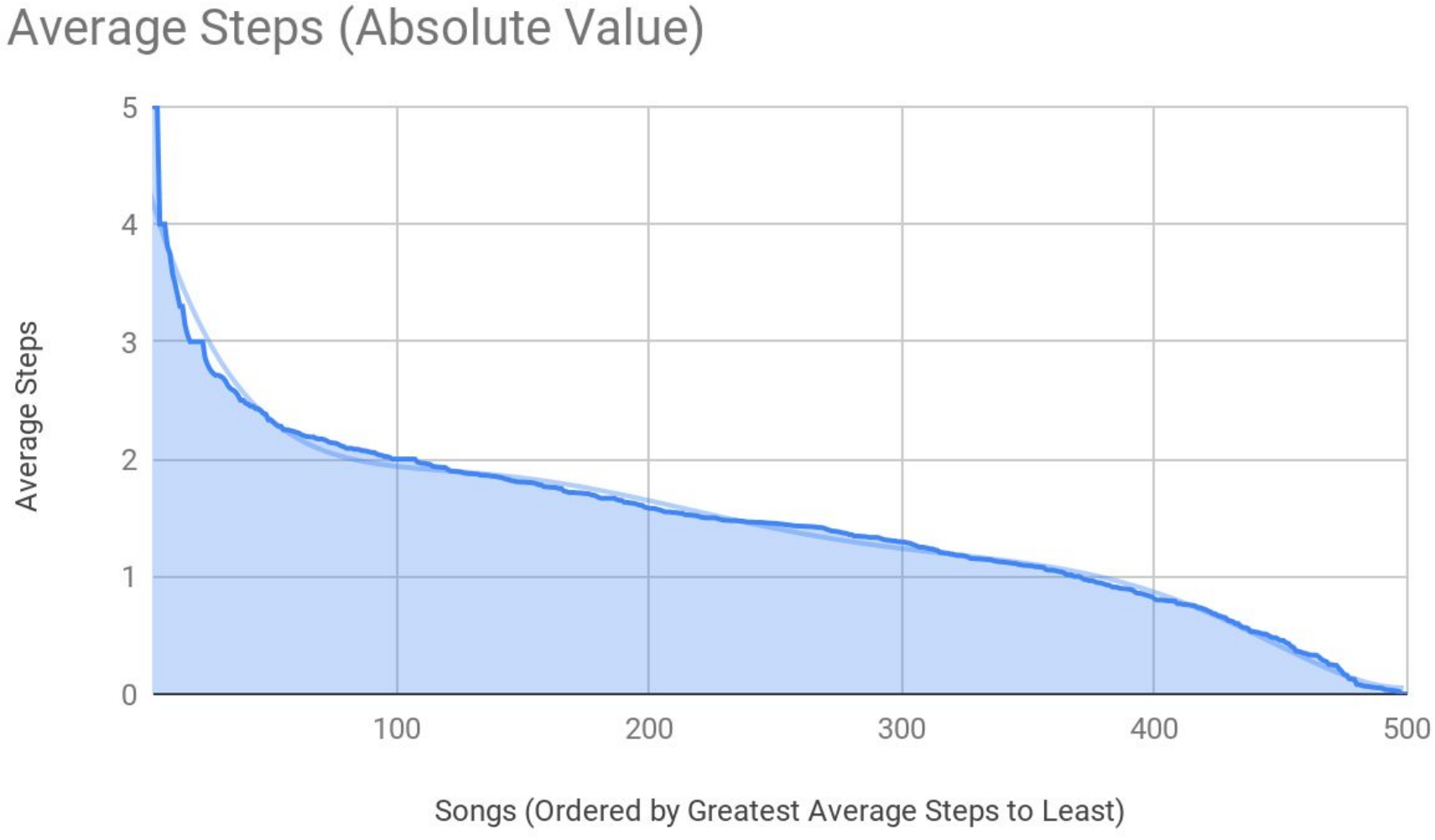
Average Steps Per Jump

The number of steps in a jump is a measure of how the pitch changed between two notes. This can be measured in two ways: including negatives (if the pitch got lower), and using absolute value. We found both. We started by finding the number of steps between the first and last notes of the song. The modes for the first four time periods was consistently one for almost all of the time periods, but in 20th Century that value, surprisingly, turned out to be about four.



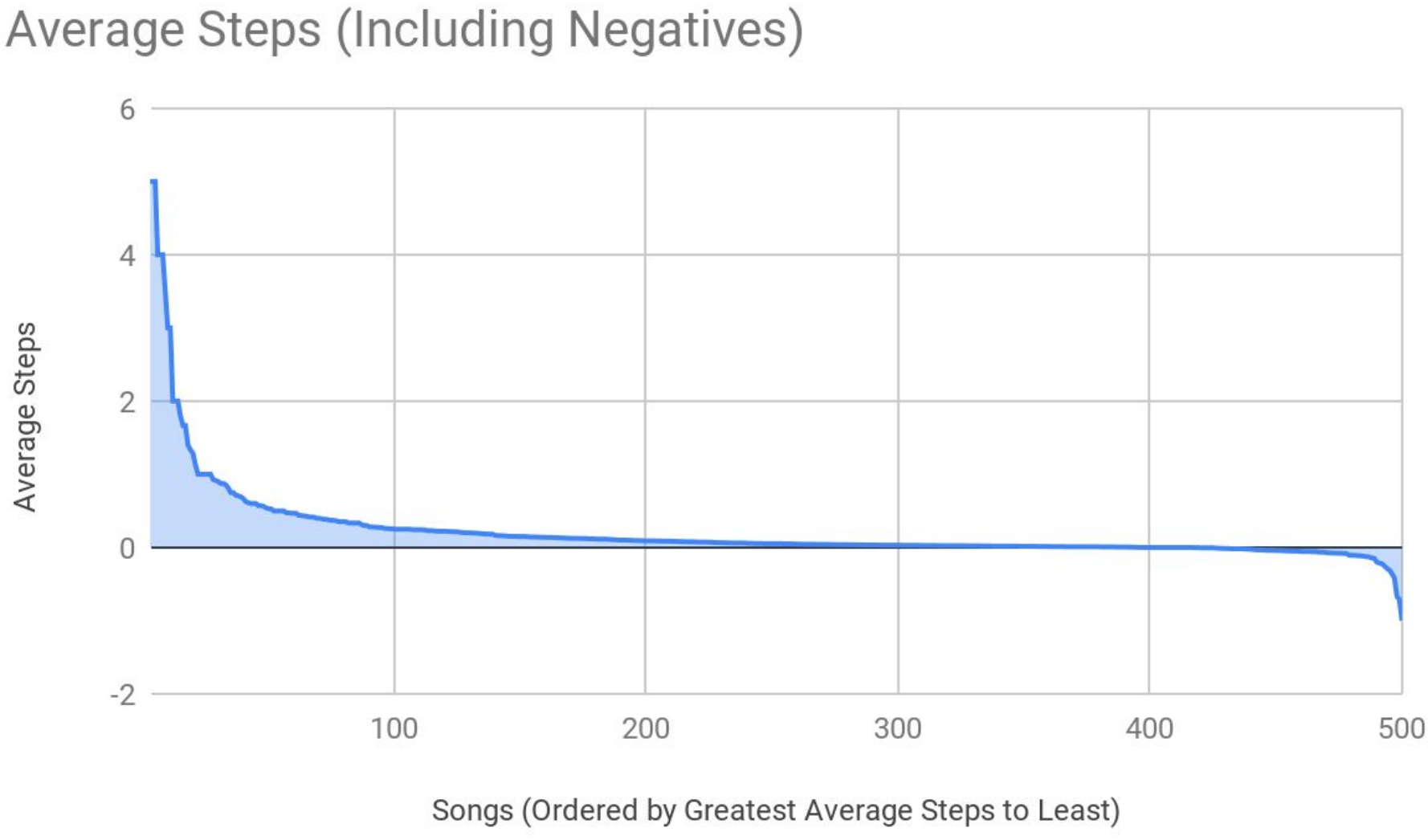
Average Steps Per Jump (Cont.)

Fig 1a (below): A graph shows absolute value with the y axis being values and the x axis being usage rank, for every song we analyzed. The curve created very accurately follows a power law until around a usage rank of 400, which is where the values begin to sink faster than would be predicted, eventually reaching a bottom of pretty much zero at around usage rank 500.



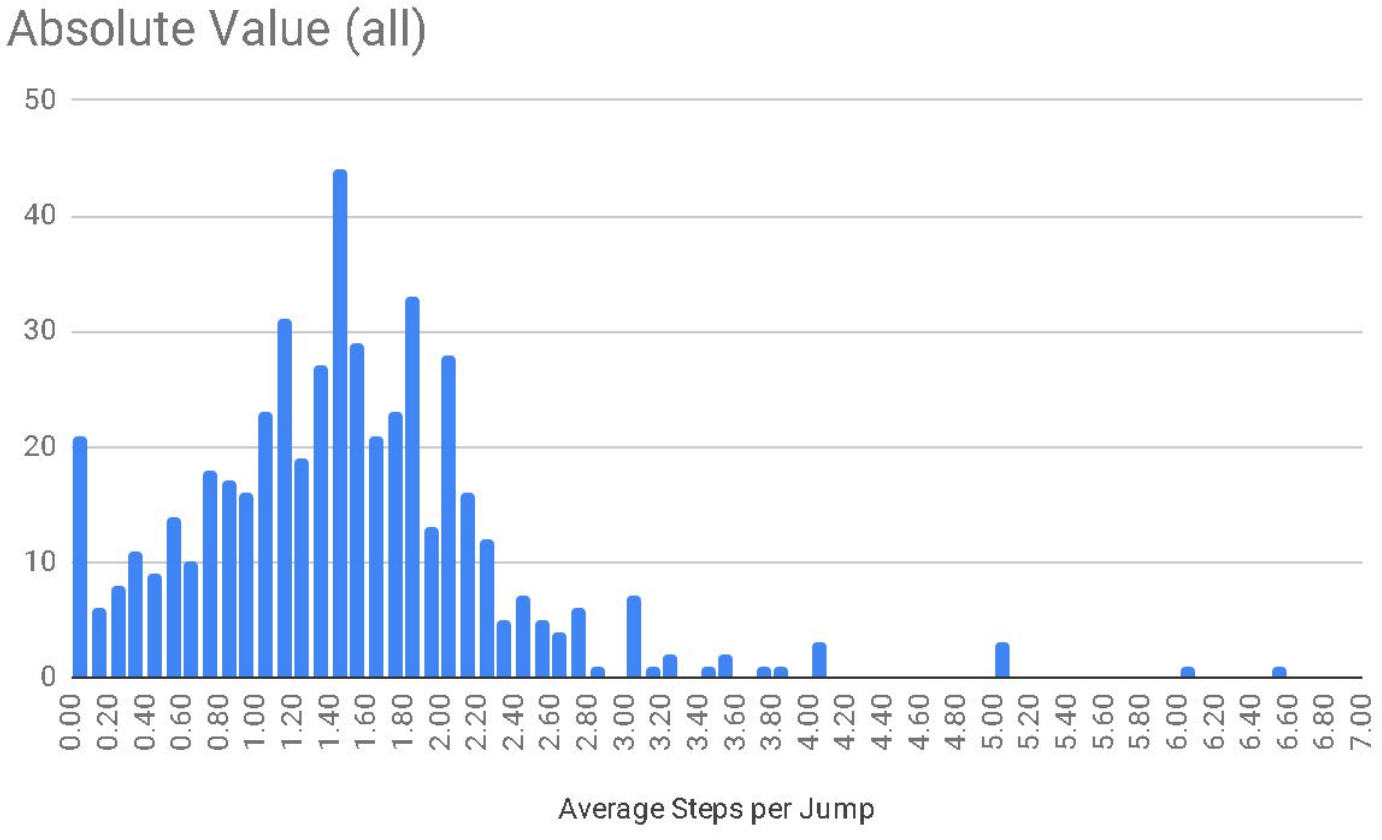
Average Steps Per Jump (Cont.)

Fig 1b (right): The same graph from above is shown again, but this time with negatives included. This graph shows a much steeper distribution, dropping to just 0.25 by usage rank 100, vs. 2 for the absolute value graph. Additionally, this graph hits zero at around usage rank 400, and then proceeds to drop into the negatives, with a minimum of -1.53 at usage rank 500.



Average Steps Per Jump (Cont.)

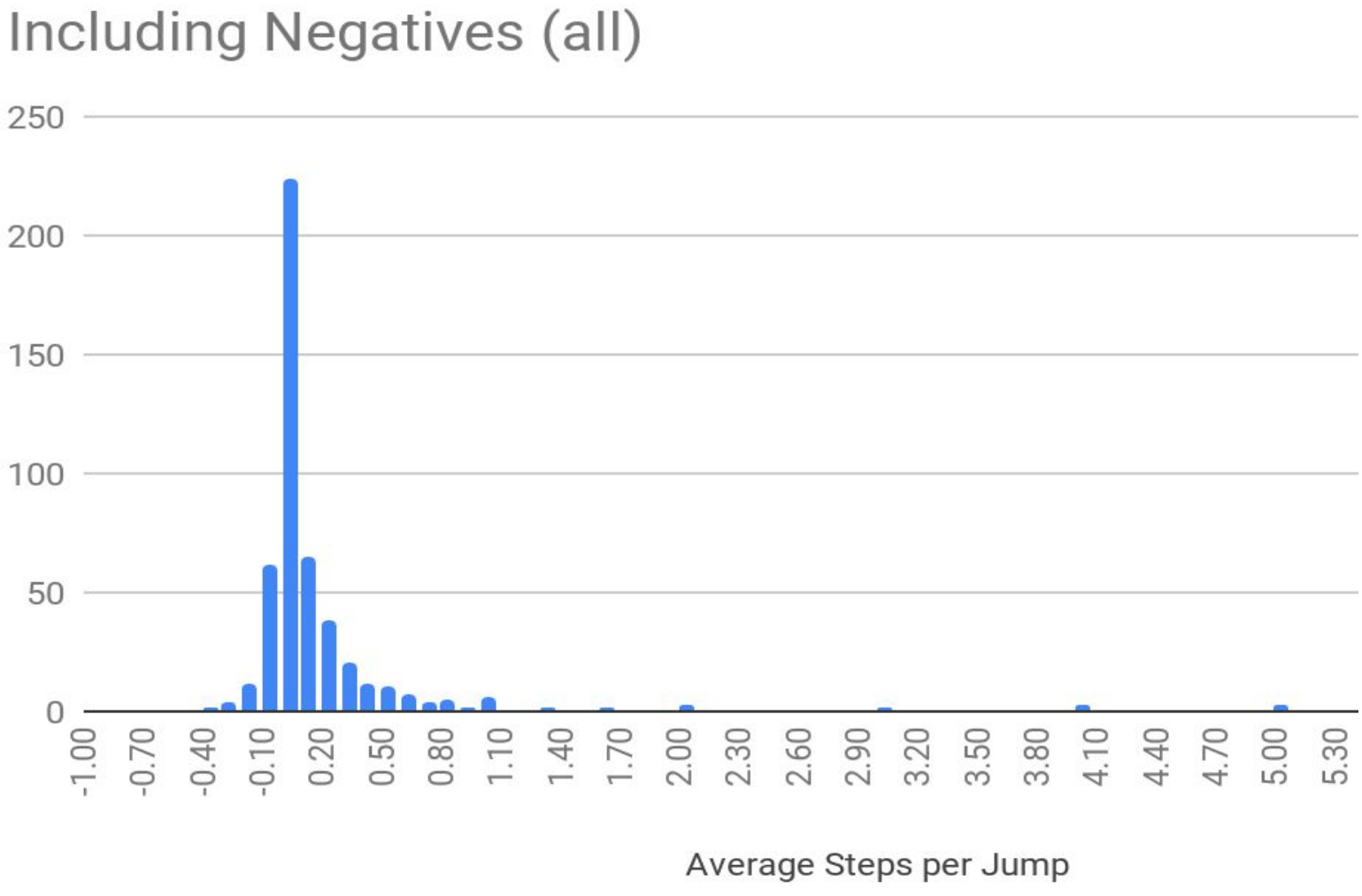
We then created a histogram of the values to gain some further insights into the distribution of the values. The first thing that jumps out is the much wider distribution of absolute value compared to negatives (which makes sense, when paired with the graphs on the previous page), although the scale is slightly warped so negatives are actually slightly further apart than portrayed in the graph. It is also worth noting that the standard deviation for absolute value is almost exactly one, with an exact value of 1.000309.



Figs 2 and 3: The histogram shows some interesting things. While absolute value looks like it may follow a normal distribution, it does not, with a skewness of 1.36 and a kurtosis of over 5. Upon closer inspection, we can see a linear tail to the left and a more power-curve-esque distribution on the right.

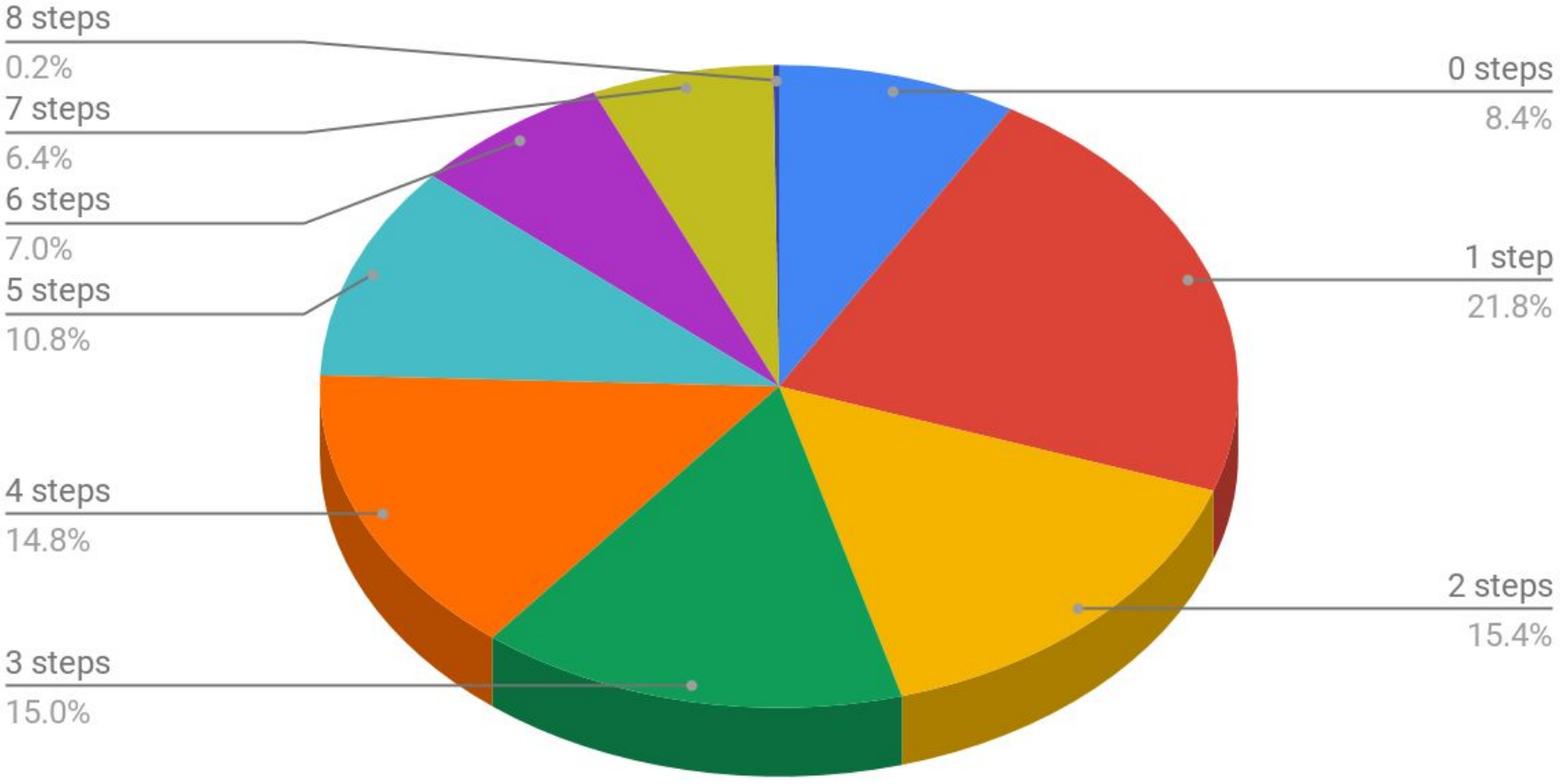
Average Steps Per Jump (Cont.)

The histogram for negatives does appear to have a larger tail on the right, creating a more gradual (although still steep) curve for the positives compared to the negatives, which are fewer and have a steeper curve compared to the positives. The negative plot actually has a slightly smaller kurtosis, with 4.29 instead of just under 5 like for absolute value.



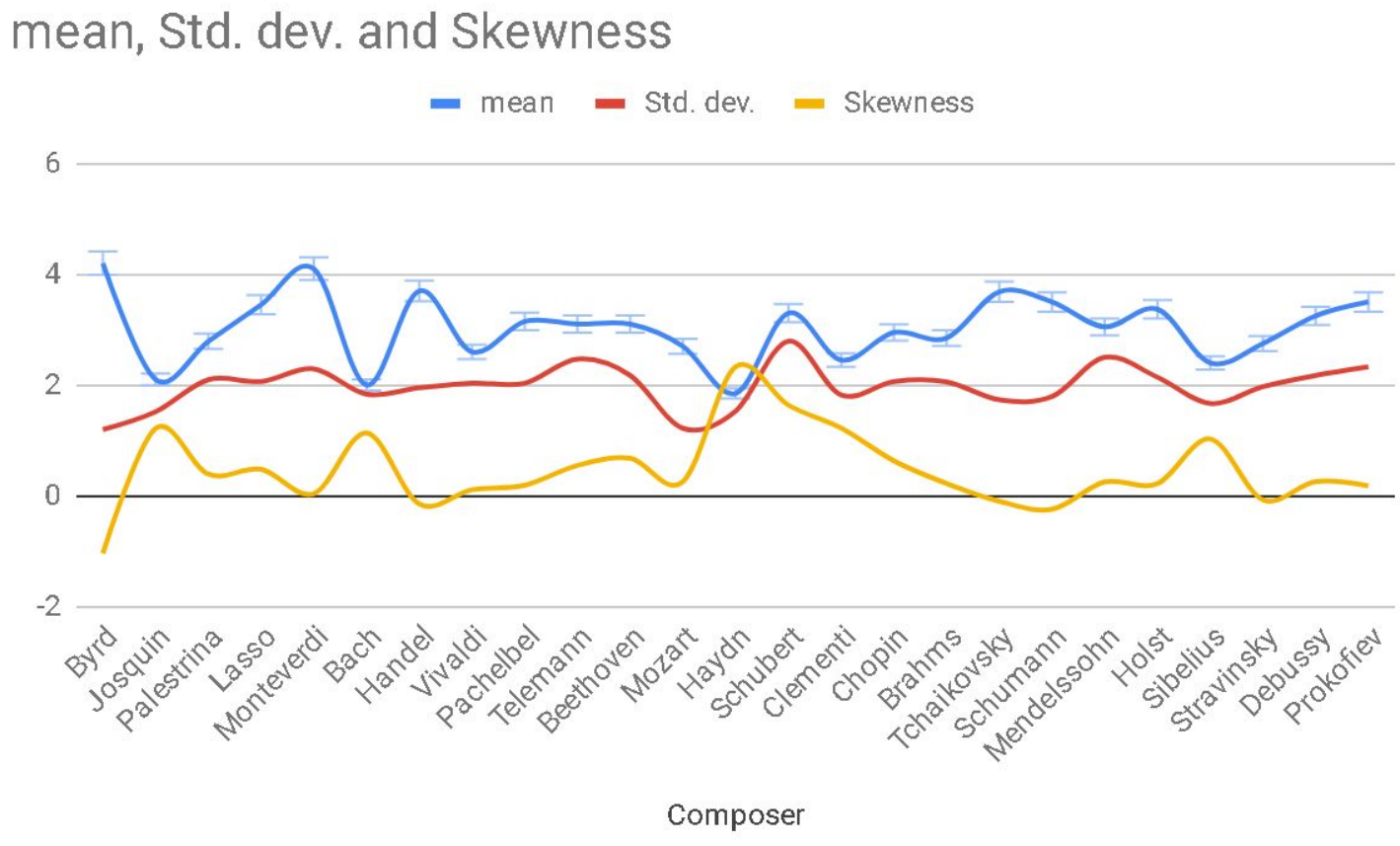
Average Steps Per Jump (Cont.)

This pie chart details exactly how many songs have what amount of difference between the first and last note.



The graph to above shows how a 1 step difference is the most common, followed by 2 steps, 3 steps, 4 steps and then 0 steps. We can see that as the steps increase, the percent decreases.

Average Steps Per Jump (Cont.)



Figs 4: The graph above illustrates several interesting things. For example, while the mean is always above the standard deviation, Bach's mean only rises slightly above the standard deviation (the skewness also spikes here). It is also worth noting that the standard deviation bears no correlation to the mean, unlike in other metrics. Additionally, Haydn is the only composer that has a skewness larger than both his mean and his standard deviation (the skewness steadily decreases from here, bottoming out at Schumann).

Scales

Scales are repeated and consistent increases or decreases in pitch.
For analyzing scales, we graphed the number of scales over the length of the scale, by time period. The results from this weren't totally unexpected, as the total number of scales went down exponentially with increasing length of the song. Additionally, Classical uses the most amount of scales for every single length of scale, while Renaissance uses the fewest number of scales (again for every length of scale).

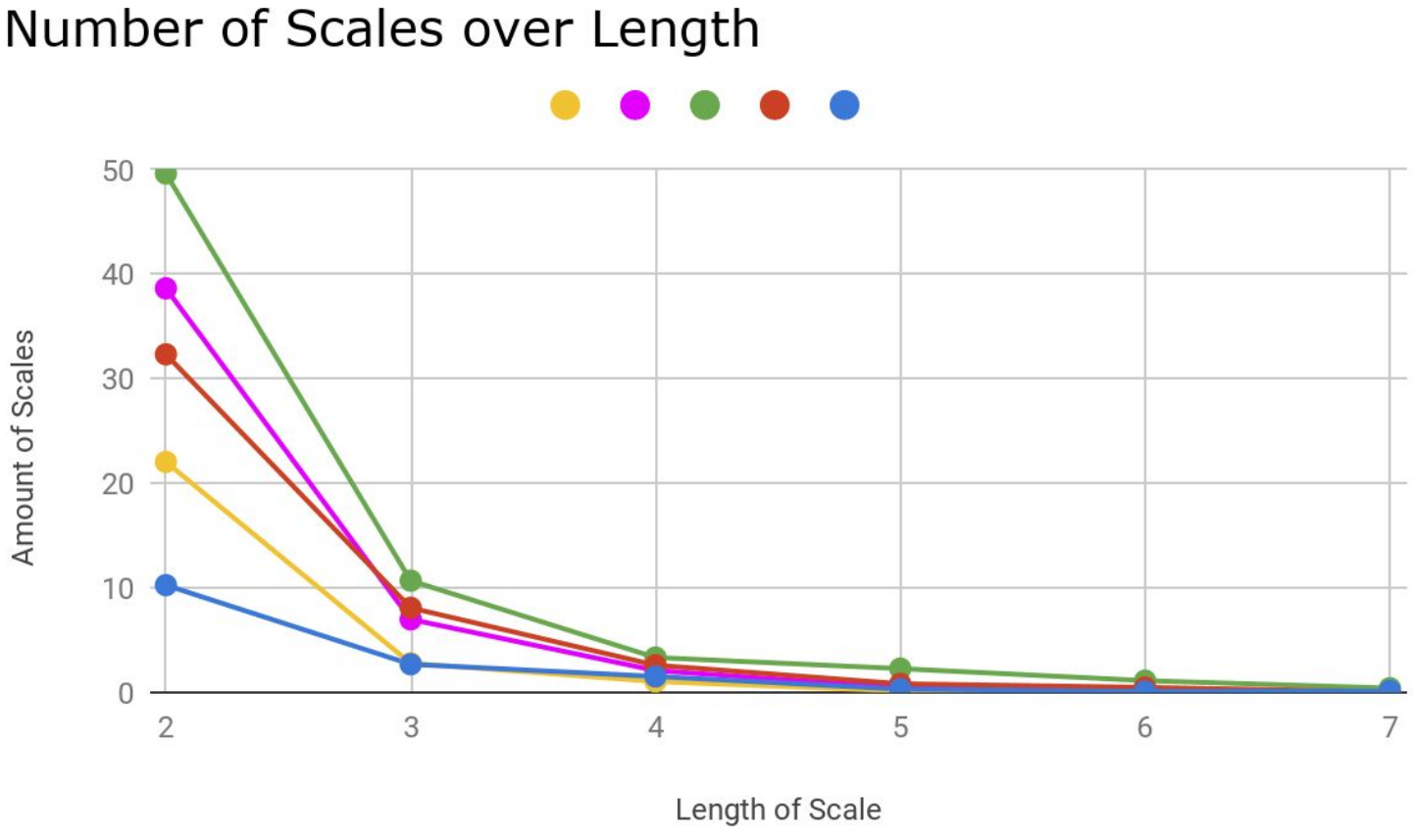
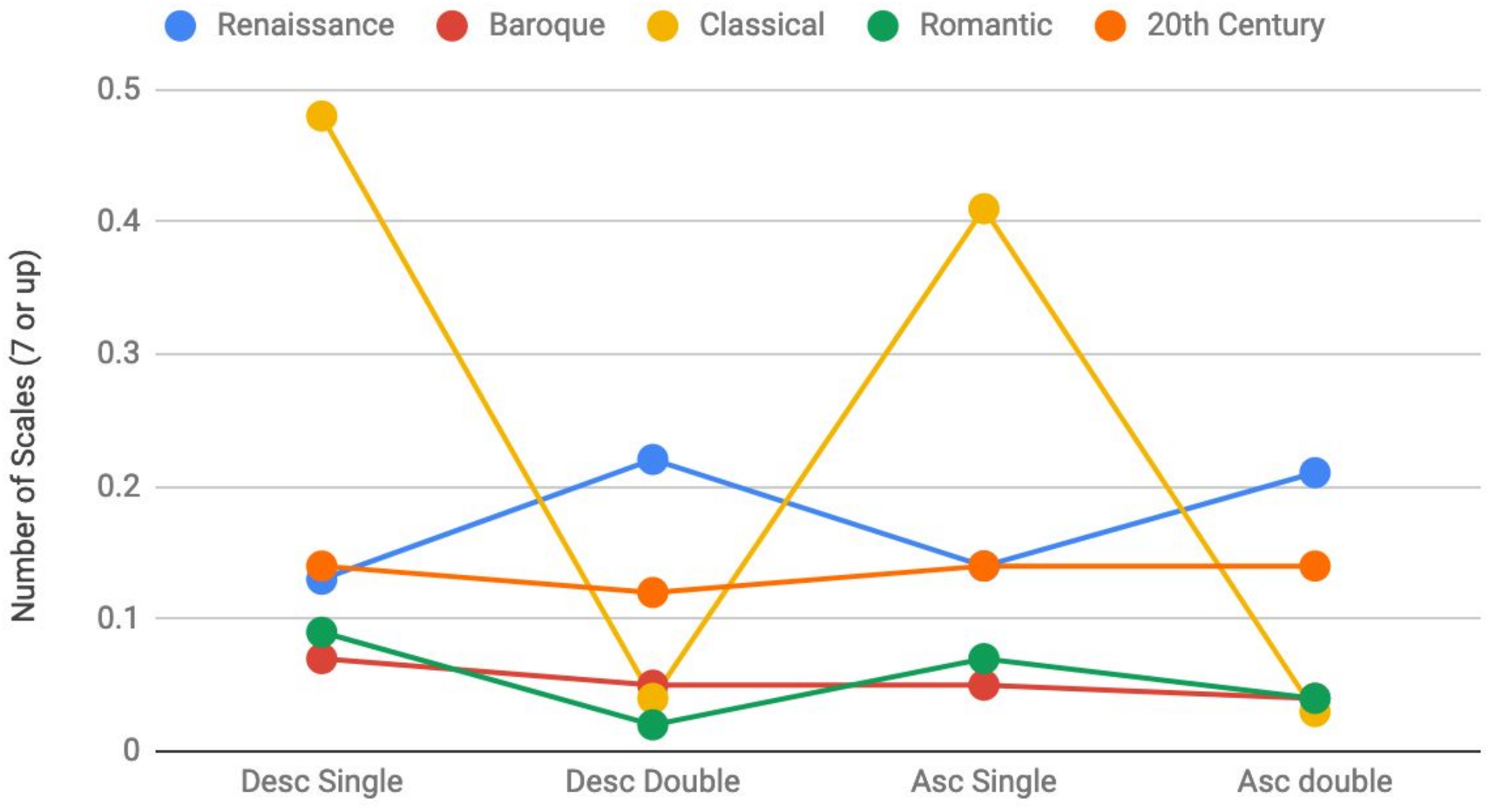


Fig. 1: A graph shows results consistent with not just other graphs in scales but also in other areas, such as Rhythmic Themes. Overall, Renaissance used far less scales compared to other time periods such as Baroque and Classical. This may be because at the time, Renaissance music was still very limited and was largely restricted to religious music. Without much experimentation, it is no wonder that our data shows such little use of embellishments by Renaissance composers.

Scales (Cont.)

The graph below show how the classical time period uses about the same amount of single step scales as the others, but uses way more double step scales.



Percent of Song in Specific Note Value

For percent of song in specific note value, we ordered the note values for all of classical music by usage rank, and then applied those usage ranks to individual composers and time periods to find equations and make bar charts, as well as analyze for Zipf's Law in different time periods.

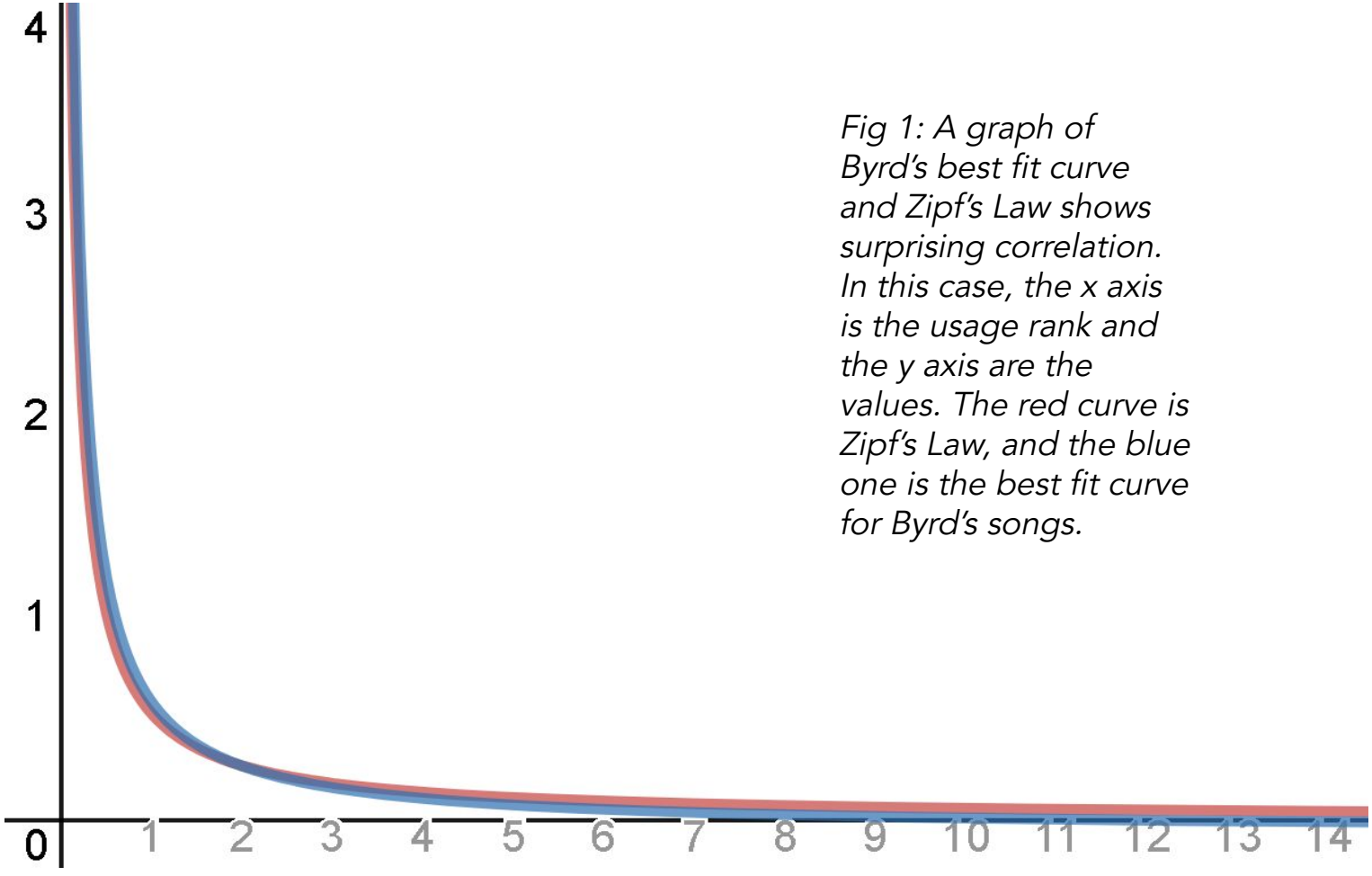
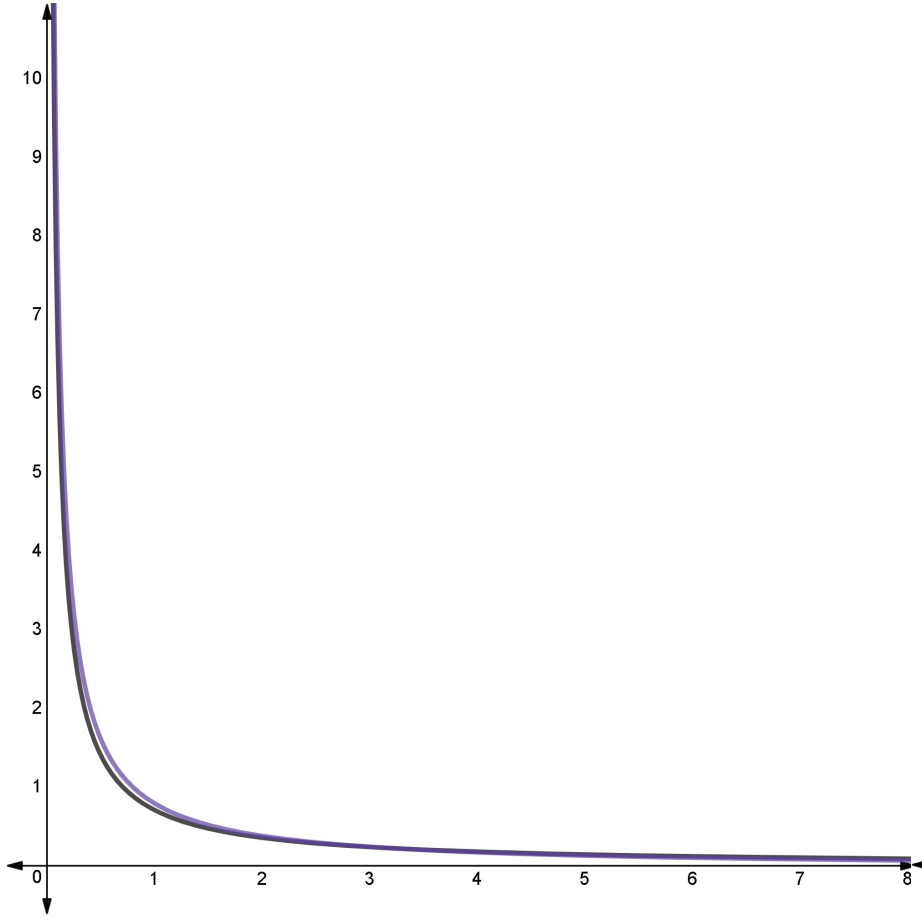
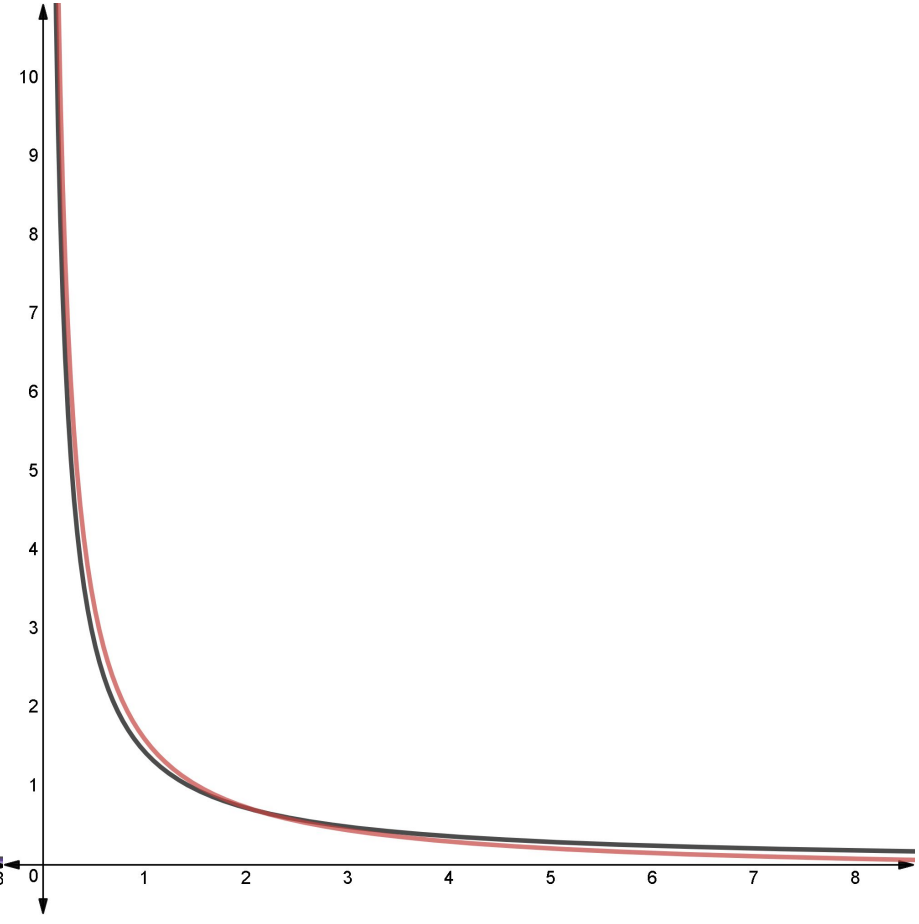


Fig 1: A graph of Byrd's best fit curve and Zipf's Law shows surprising correlation. In this case, the x axis is the usage rank and the y axis are the values. The red curve is Zipf's Law, and the blue one is the best fit curve for Byrd's songs.

Percent of Song in Specific Note Value (Cont.)

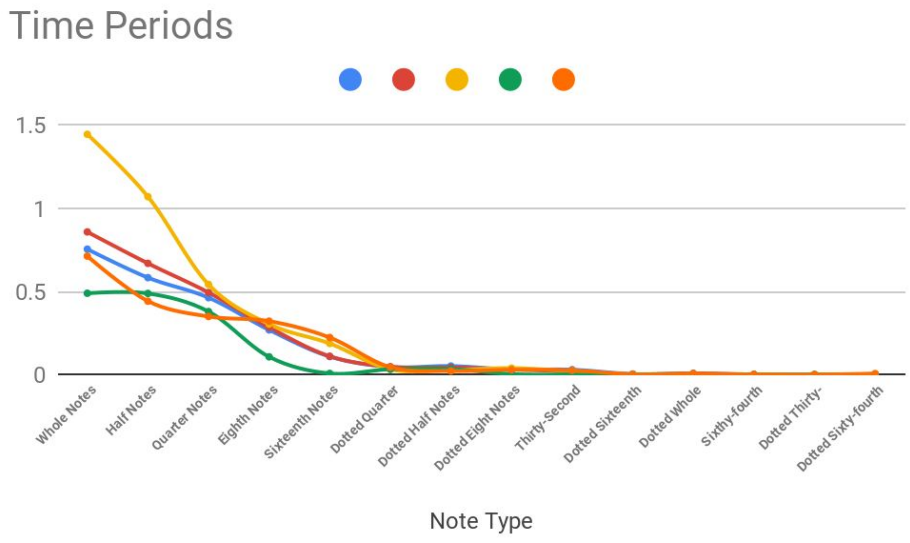
Pictured here are two more graphs for the best fit equations vs. the values predicted by Zipf's Law. As you can see, the curves correlate well.





Figs 2 and 3: The graph on the left is of the Baroque time period. The black curve is Zipf's Law and the purple one is the best fit curve for our data. The graph to the right is also a time period, but this time Classical. The black curve is Zipf's Law, and the red one the Classical best fit curve. Out of all 5 time periods and classical music, only Baroque and Classical fit Zipf's Law.

Percent of Song in Specific Note Value (Cont.)



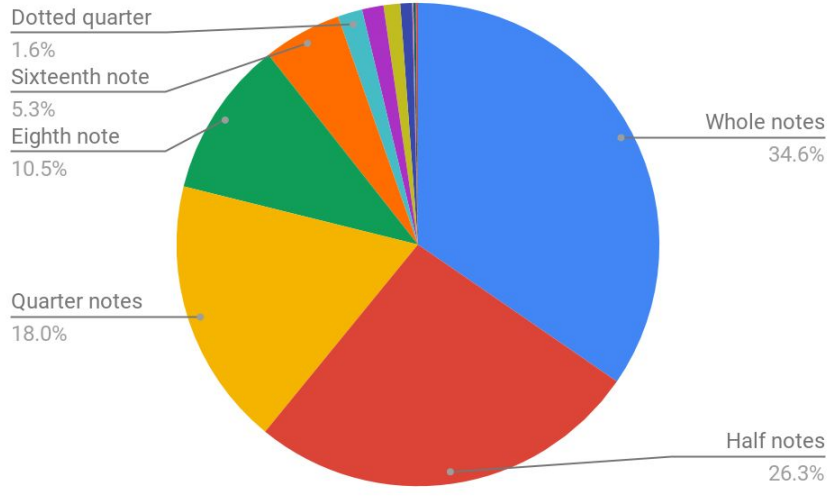


Fig 4: A smooth line chart of all the time periods show the percentages of the average song in their time period, by note type. The notes are arranged in descending order for all, and the percentages, while they don't add up to 100%, are still proportional. It can be seen here that whole notes and, to a lesser extent, half notes, were used much more often in Classical than in any other time period. While the other time periods always use less half notes, Romantic actually uses more half notes.
Fig 5: A pie chart of all of classical music shows the makeup of an average melody. Whole notes make up 34.6% of the song, and the percentage of the song in a note steadily decreases as the length of the note decreases up to sixteenth notes (decreasing by around 8% until eighth notes, no less). This pie chart also shows evidence of the 80/20 rule. Taking the top 3 note values, whole notes, half notes, and quarter notes, makes up 78.9% of the entire song. And since there are 14 note values we analyzed for, the whole notes, half notes, and quarter notes make up 3/14=21% of the notes, basically 20%.

Percent of Song in Specific Note Value (Cont.)

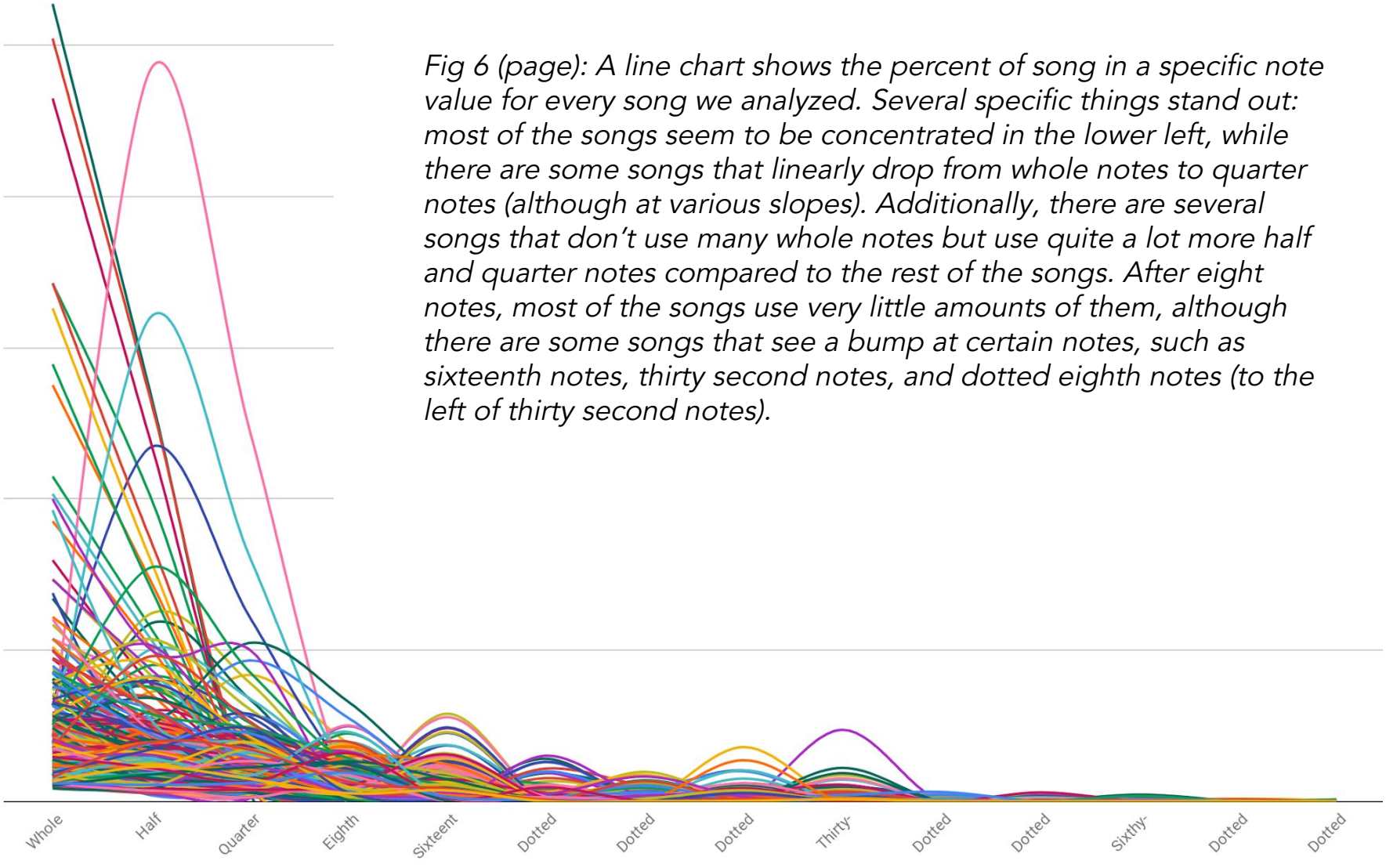


Fig 6 (page): A line chart shows the percent of song in a specific note value for every song we analyzed. Several specific things stand out: most of the songs seem to be concentrated in the lower left, while there are some songs that linearly drop from whole notes to quarter notes (although at various slopes). Additionally, there are several songs that don't use many whole notes but use quite a lot more half and quarter notes compared to the rest of the songs. After eight notes, most of the songs use very little amounts of them, although there are some songs that see a bump at certain notes, such as sixteenth notes, thirty second notes, and dotted eighth notes (to the left of thirty second notes).

Section 3:

Formula

The Formula

Another one of our project's goals was to come up with a way to quantify how close, or similar, two songs, time periods, or composers are to each other.

Below is the entire formula we wrote that compares two songs, composers, or time periods. It outputs a number between zero and one, showing the correlation between the two things being compared. Though it may look quite complicated, it's actually somewhat simple, and we'll break it down.

$$\sum_{i=1}^{n} \left(\frac{r_i}{\sum_{k=1}^{n} r_k} \right) \left(1 - \frac{|d_i - a_i|}{\sqrt{(a_i)(d_i)}} \right)$$

Meaning

As shown below, this formula requires three lists, or inputs: *a*, *d*, and *r* values. The *a* and *d* values are the measurements of each data set. For example, in our measurements the first object of the set of *a* values would be the average note value. The *d* value would then also be the corresponding average note value for the song you are trying to compare. The set of r values contains the correlation coefficient for each metric.

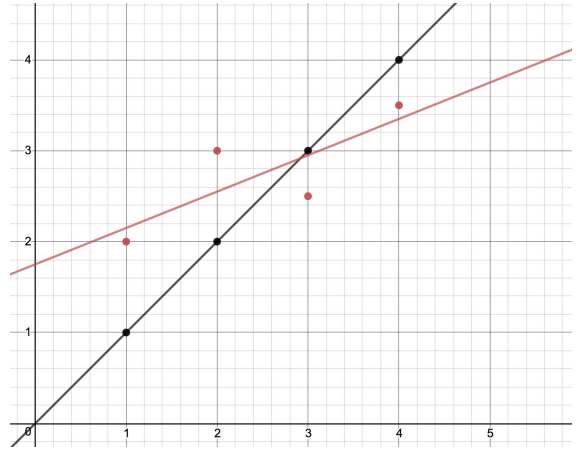
Metric	m_1	m_2	$m_{...}$	m_n
R Values	r_1	r_2	$r_{...}$	r_n
A Values	a_1	a_2	$a_{...}$	a_n
D Values	d_1	d_2	$d_{...}$	d_n

Next, let's have a look at the first factor of the summation, shown here bolded.

$$\sum_{i=1}^{n} \left(\frac{r_i}{\sum_{k=1}^{n} r_k} \right) \left(1 - \frac{|d_i - a_i|}{\sqrt{(a_i)(d_i)}} \right)$$

In this formula, we use correlation coefficients for weighting the averages of our metrics. The reason for this is that if the data follows a trend, then it is more likely to have a powerful influence on the song.

The correlation coefficient is a measure of how well an equation can describe the relationship between a set of x values to their corresponding y values. In the graph to the right, the red set of points have a weaker correlation, while the black ones have a strong correlation.



The first term takes the *r* value of the current metric, and turns it into a fraction out of the sum of all of the *r* values. This way, when multiplying by each of our values from the next factor (which are all less than one), the sum of them will always be less than 1.

The next factor of the summation deals with the correlation between the measurement of each song. What we originally had for this factor was:

$$(- \left| \frac{a_i}{d_i} - 1 \right| + 1)$$

The absolute value function that surrounds a/d is to ensure that it is less than 1, but still the same distance from 1. It takes the value's distance from 1, and subtracts it from 1. The problem with this was that the a/d is not equal to d/a, meaning that comparing Baroque to Renaissance would output a different value from Renaissance to Baroque. We realized our flaw and corrected it by following the process of the old method with sample values of 35 and 40.

This old function would give us these two different values:
35/40 40/35
1-5/40 1-5/35

We realized that the numerators (5 in this case) will always be the same, and that if we average the denominators, we will be accounting for both of these values. We decided to use geometric mean instead of arithmetic mean because our data is in the form of a ratio, but in different units. The central tendency of this kind of data is generally agreed to be best represented by geometric mean.

Other Applications

This formula is not, however, exclusive to musical analysis. It is essentially a new type of statistical test. It is similar to a t-Test in the way that it compares the similarity of two datasets, and is similar to correlation coefficients in the way that it outputs a value within a certain range that came from two variables. It weights different metrics based on how important (and correlated) the metrics are, thus accurately comparing two different datasets, and can be extended to as many metrics as needed, provided the two (or more) data sets have the same number of corresponding data points.

This can apply to all sorts of real-world situations. For example, this can be used for comparing the grades of two students, while being able to give greater importance to assessments and projects than regular assignments.

Part B:

Analysis

After designing a functioning formula, we now applied it by evaluating it on the data we had for every composer and time period to every other composer and time period. This allowed us to see how closely each composer and time period were correlated, and allowed us to test earlier theories about Renaissance and transitional composers. Also, we were able to measure and graph change in correlation over time, which allows us to predict how correlated time periods will be to more modern music, and even music that has yet to be written. The following is presentation of the data we collected after evaluating the formula.

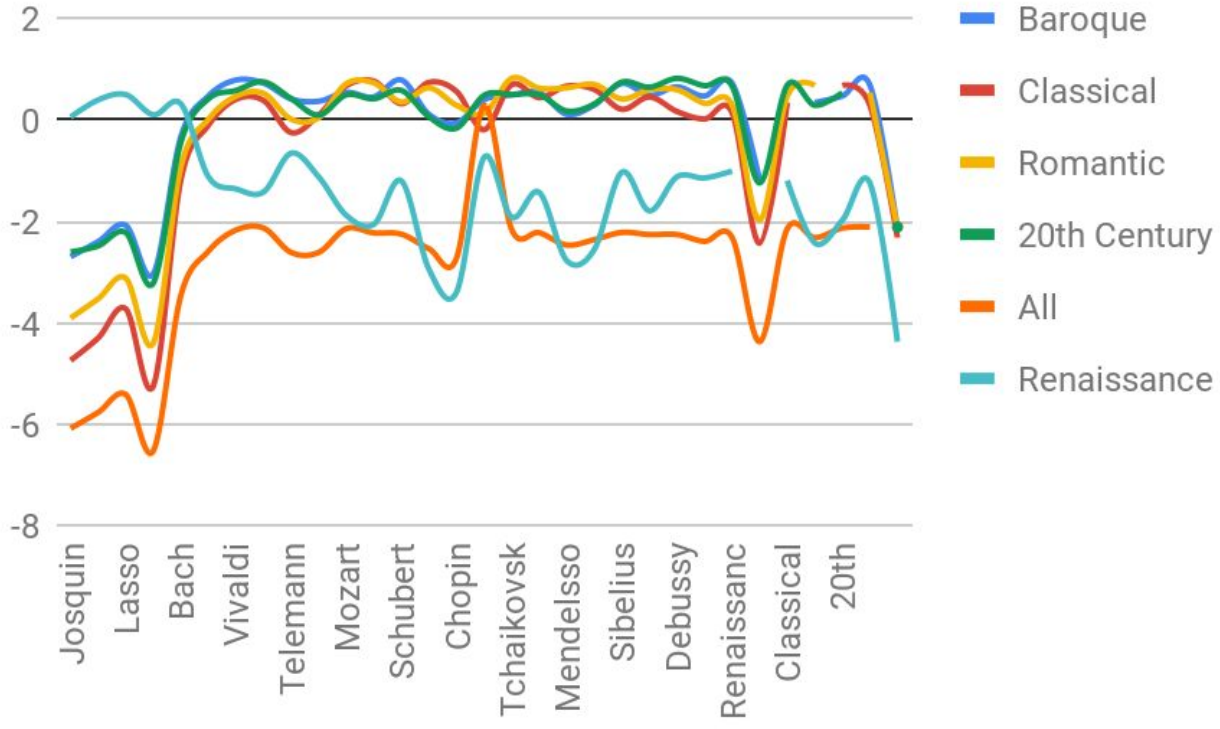
Charts

We first compared the time periods and classical music to each other by doing t-tests on the correlations to every category (composers, time periods, and classical music) and seeing if they were fundamentally different. Classical music is different from every single time period, and Renaissance is different from all of the other time periods. However, the other four time periods have similar correlations to the categories, to the point where null hypothesis cannot be rejected and thus makes the correlations not inherently different.

		P Values:	Null Hypothesis Rejected?
All	Ren	0.000026392	Yes
All	Bar	0	Yes
All	Cla	0.00000123	Yes
All	Rom	0	Yes
All	20th	0	Yes
Ren	Bar	0.0000468	Yes
Ren	Cla	0.048475	Yes
Ren	Rom	0.00646	Yes
Ren	20th	0.000676	Yes
Bar	Cla	0.23763	No
Bar	Rom	0.47662	No
Bar	20th	0.67867	No
Cla	Rom	0.62775	No
Cla	20th	0.40997	No
Rom	20th	0.74011	No

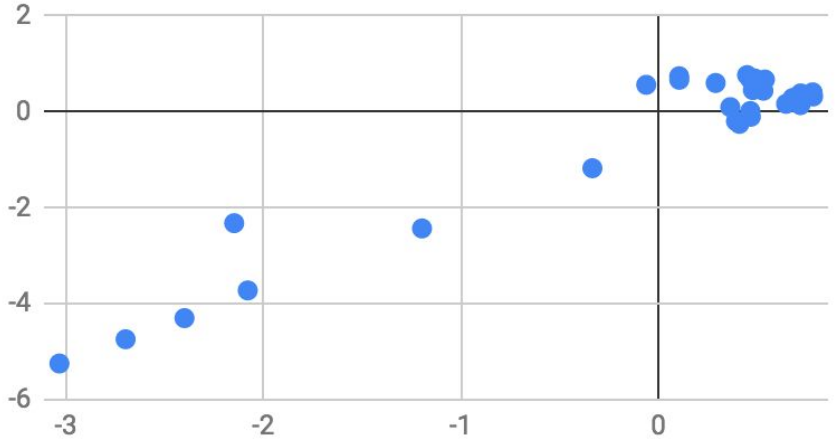
Charts (Cont.)

We also graphed the correlations on a smooth line graph for each time period (excluding Renaissance) and all (the orange line). You can see a striking correlation with the later four time periods, especially with the Renaissance composers. Where Lasso is less correlated with Baroque, it is also less correlated with the later time periods as well. Additionally, Monteverdi has a high correlation to the non-Renaissance time periods (relative to the other Renaissance composers) and is on par with later composers that are actually part of those time periods. For this reason, Monteverdi is a transition composer. Another interesting results is the surprising correlation of Chopin with all of classical music, despite no noticeable change in correlation to the non-Renaissance time periods. This is because Chopin is surprisingly correlated with Renaissance, a feat not achieved by any of the non-Renaissance composers.

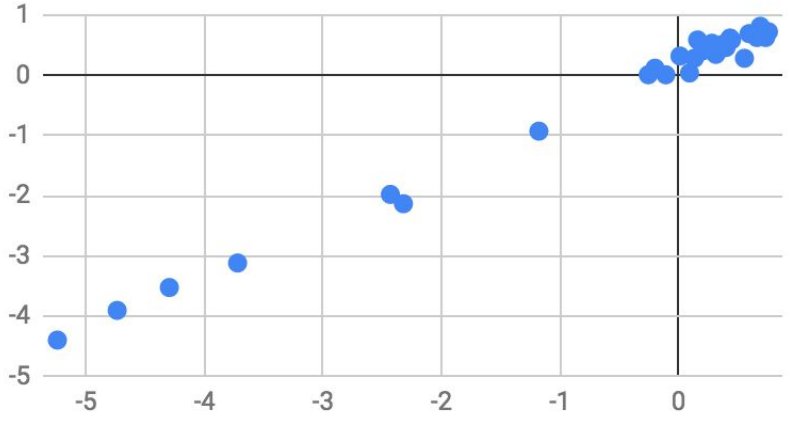


Charts (Cont.)

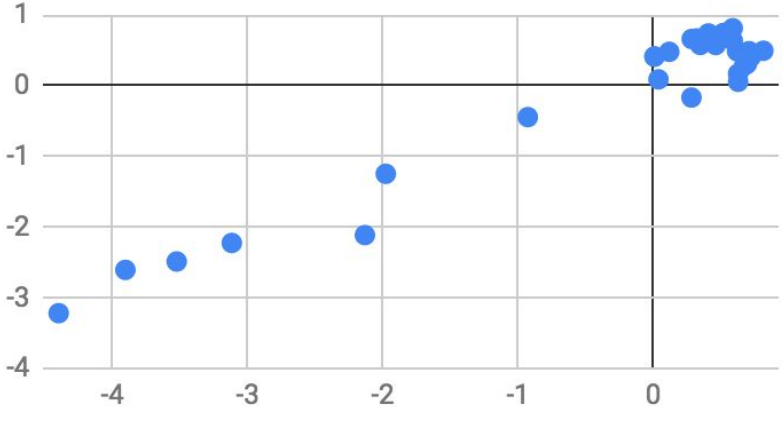
Classical to Baroque



Romantic to Classical

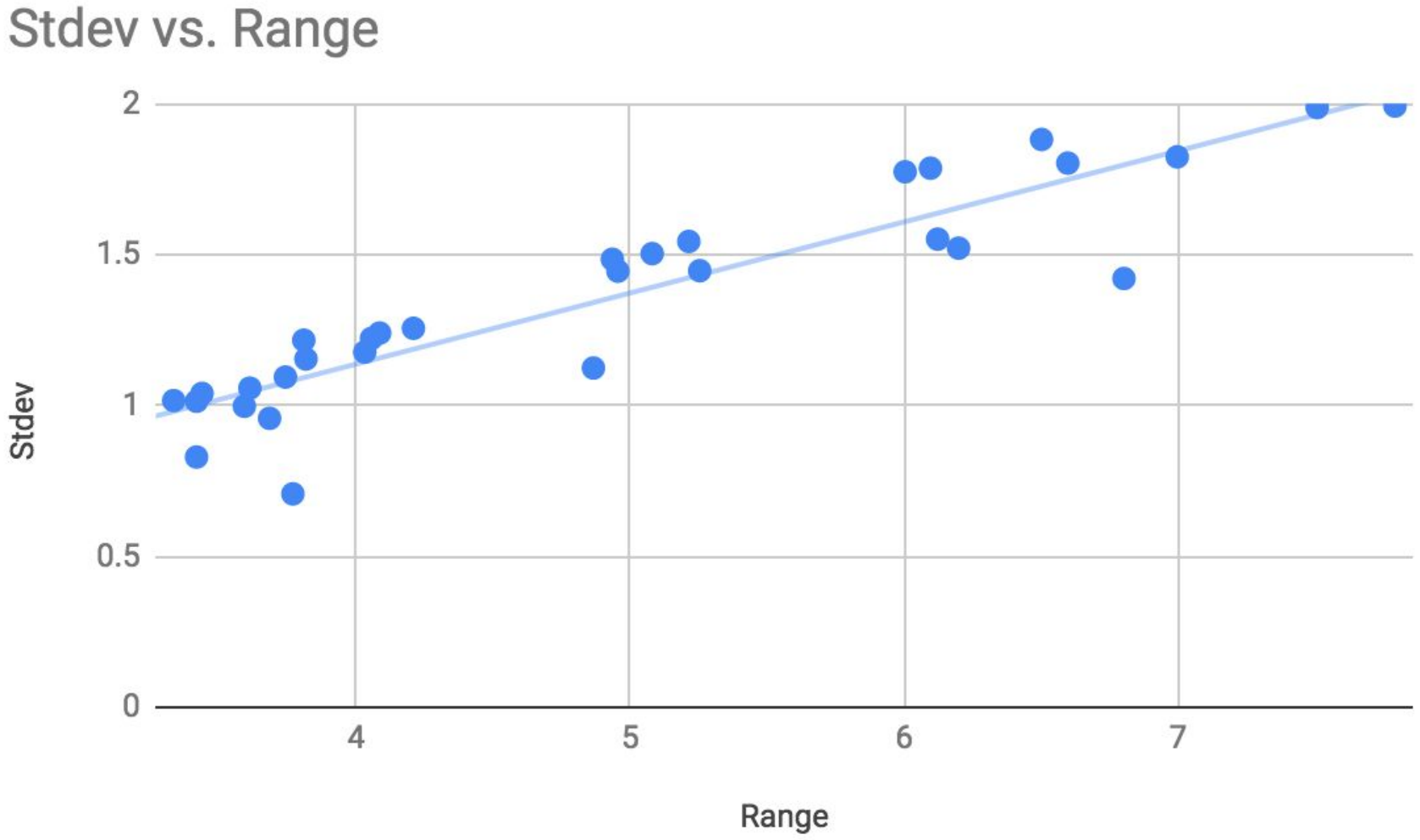


20th Century to Romantic



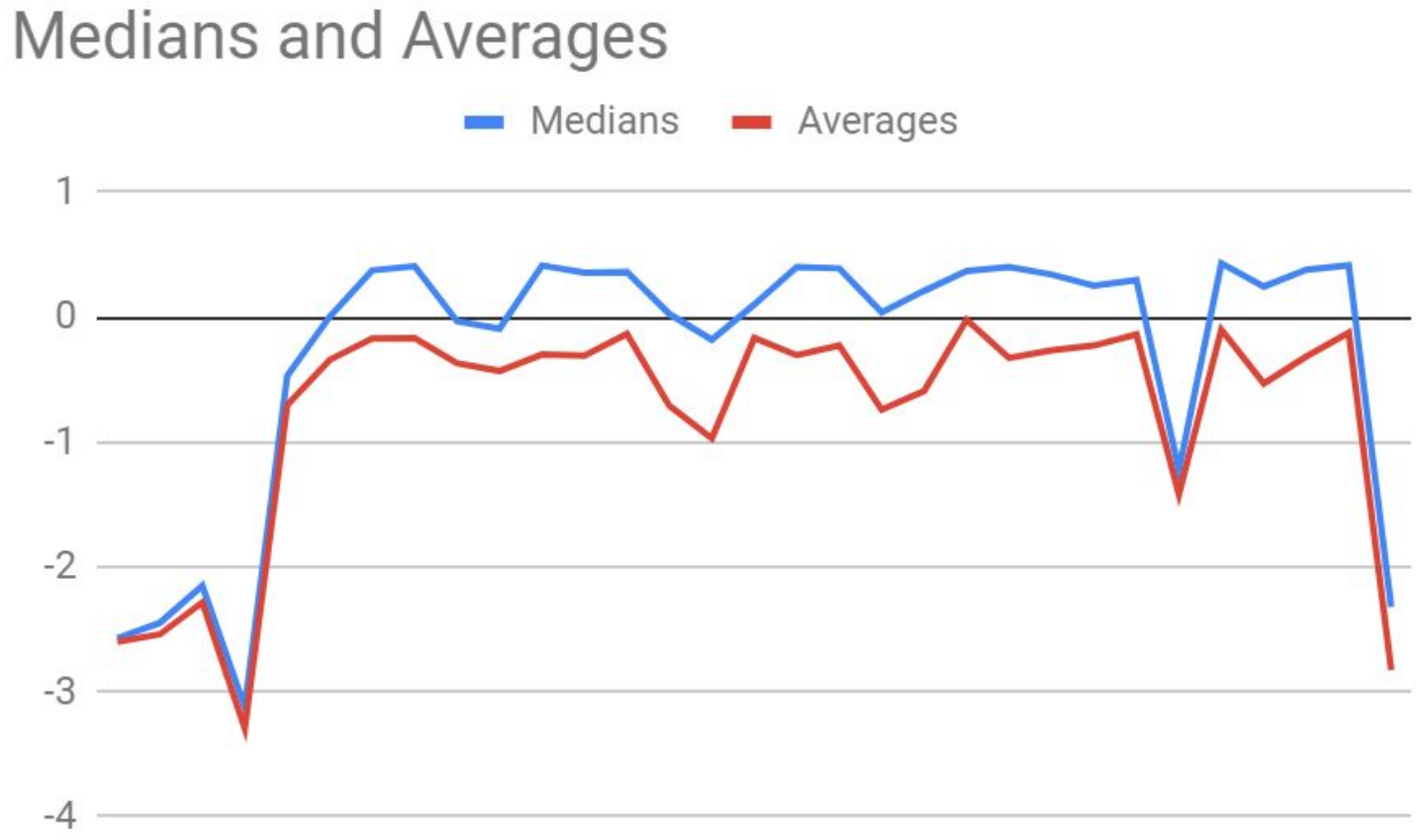
We then compared the non-Renaissance time periods to each other with scatter plots to see if the time periods followed a linear equation. It is safe to conclude that they do, with most of their points concentrated in the 0-1 range (which were the non-Renaissance composers). However, of the Renaissance composers, they follow a linear equation as well, thus showing a high correlation between the correlations of the different non-Renaissance time periods.

Charts (Cont.)



*A scatter plot of the standard deviations and ranges for the correlations for each category shows a high correlation to a linear equation, although perhaps not as high as in the scatter plots on the previous page. The data points have a r correlation coefficient of 0.919546 and the equation that best fits this line is y=0.19541174 + 0.235492092 * x.*

Charts (Cont.)



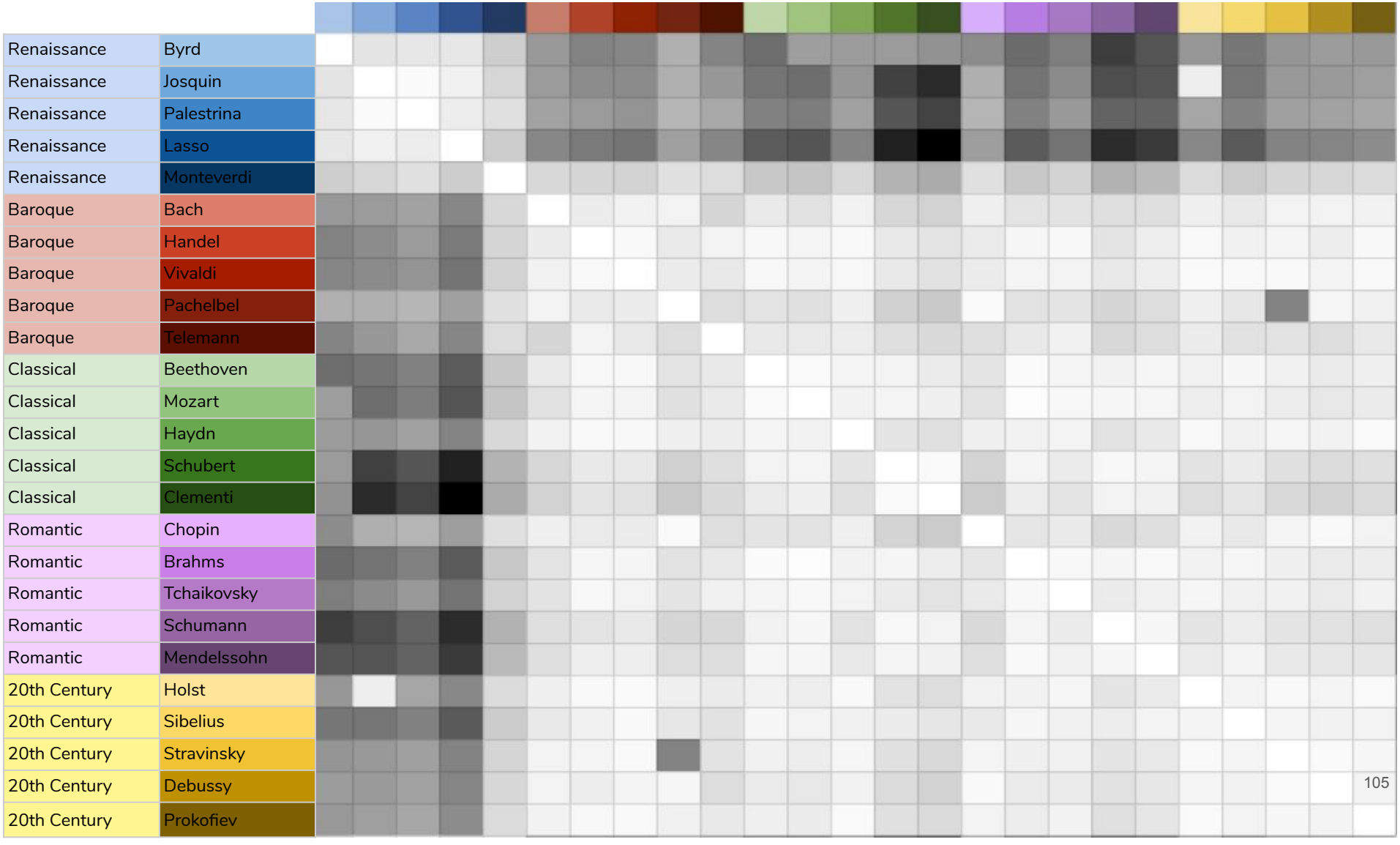
For each composer, time period, and then all, we took the average and median of the correlations and compared them on this line graph. You can see several interesting things. For example, the medians and averages for the Renaissance composers are closely correlated, while beginning at Baroque, the medians are consistently higher than the averages, meaning they were correlated but not completely correlated. This is because of Renaissance, which had lower correlations to the non-Renaissance composers and thus affected the average more than the median, causing the disparity that shows up in this graph.

Heatmaps

Heatmaps are essentially just a visual way of representing a grid of data. In our case, it allows us to quickly see which composers or time periods are least and most correlated in a visual manner. In our case, we have a spreadsheet comparing each composer to every other composer. The darker the cell, the less correlation between the two values. The lighter, the more correlation.

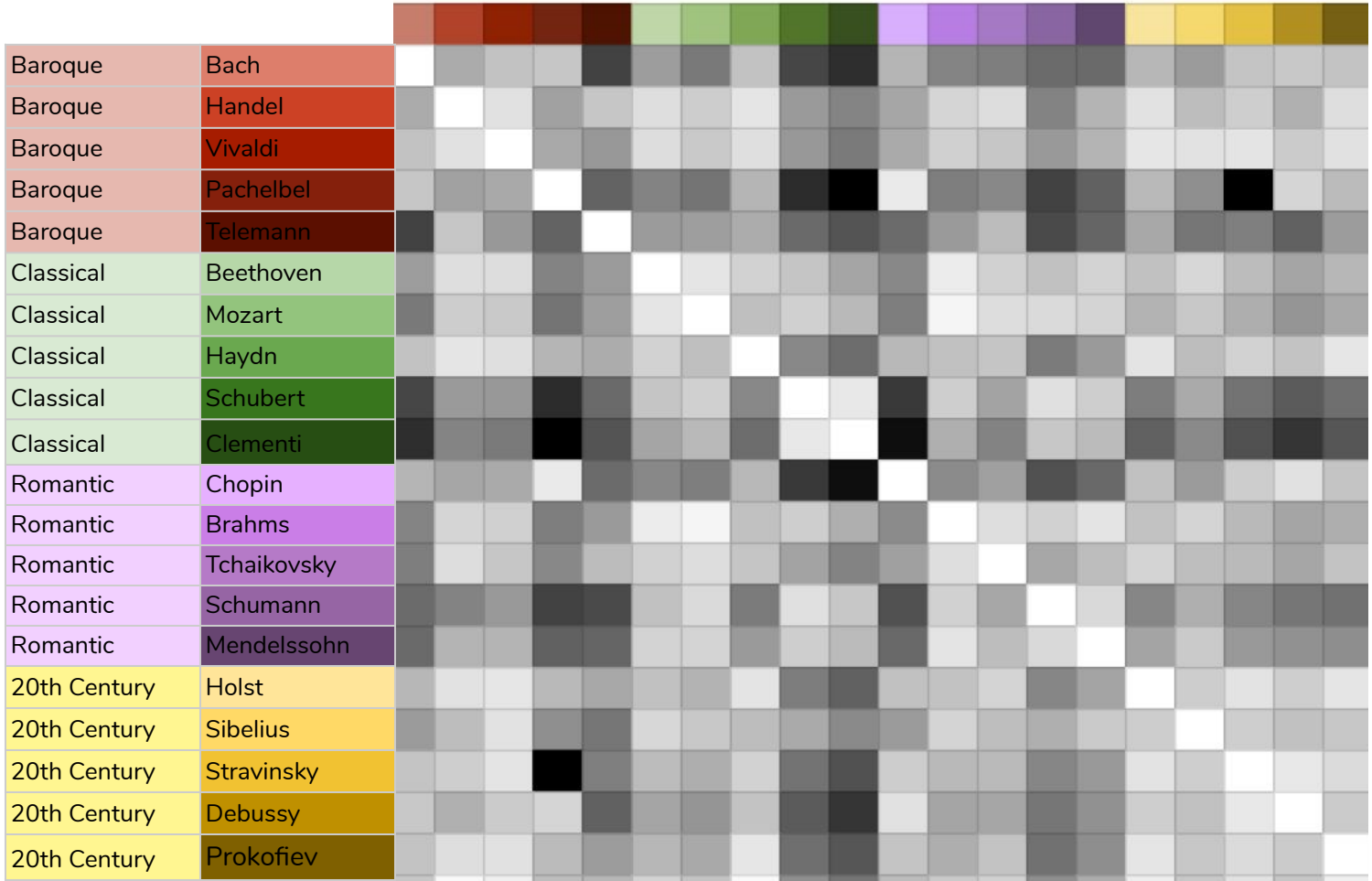
Heatmaps - Every Value

Fig 1: This heat map shows composer's correlation with every other composer. As we can see from this, there is very little correlation with Renaissance (both composers and time period) to anything else. However, we can see that Monteverdi has considerable more correlation to other composers and time periods. This could be due to the fact that he is a transitional composer.

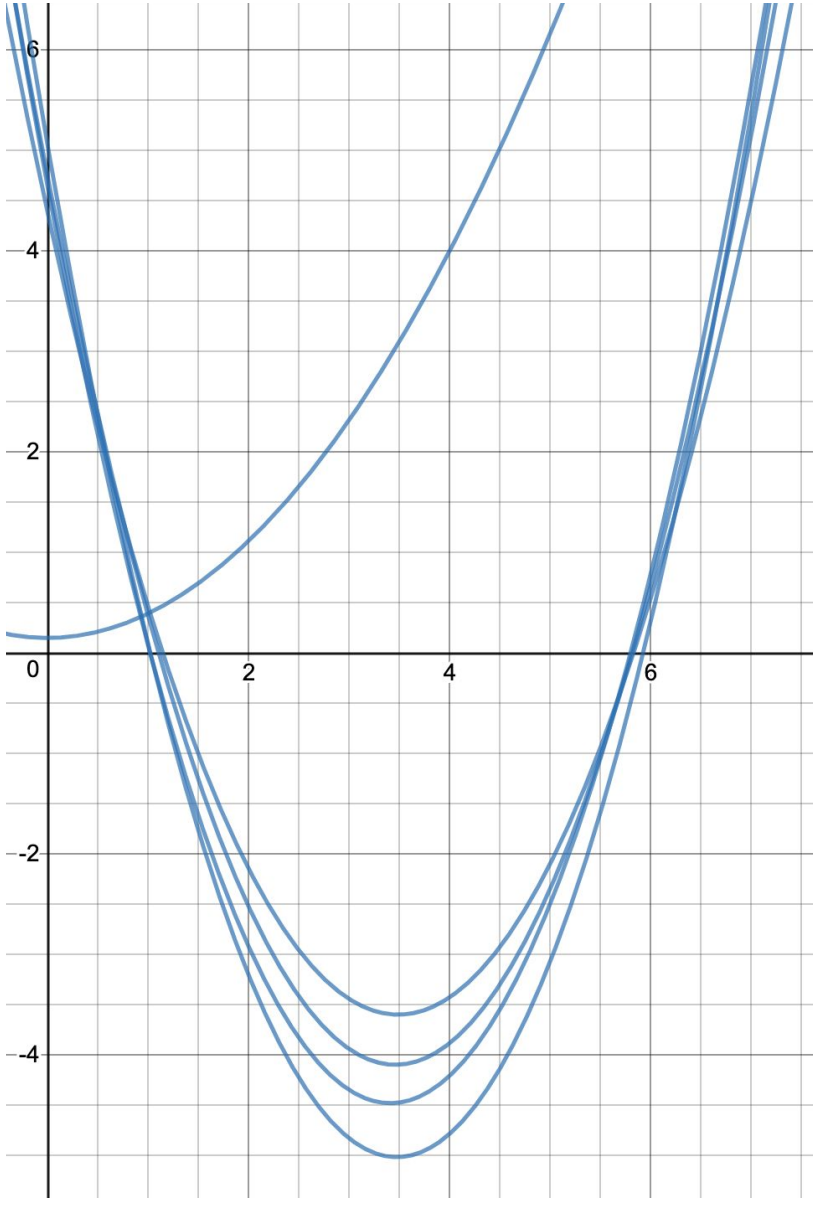


Heatmaps - Excluding Renaissance

Fig 2: Through all our analysis, we have found Renaissance somewhat strange. From strange curves to being well outside the standard deviation, we have constantly found it to be an outlier. By excluding Renaissance from our heatmap, we can more easily see correlation, as the point where a cell turns black is reduced. From here, we can see that Chopin has little correlation with major classical composers, and that Stravinsky and Pachelbel clearly have nothing to do with each other. We also notice a strange correlation between Chopin and Pachelbel, and a lack of correlation between both Schubert and Clementi to anything else.

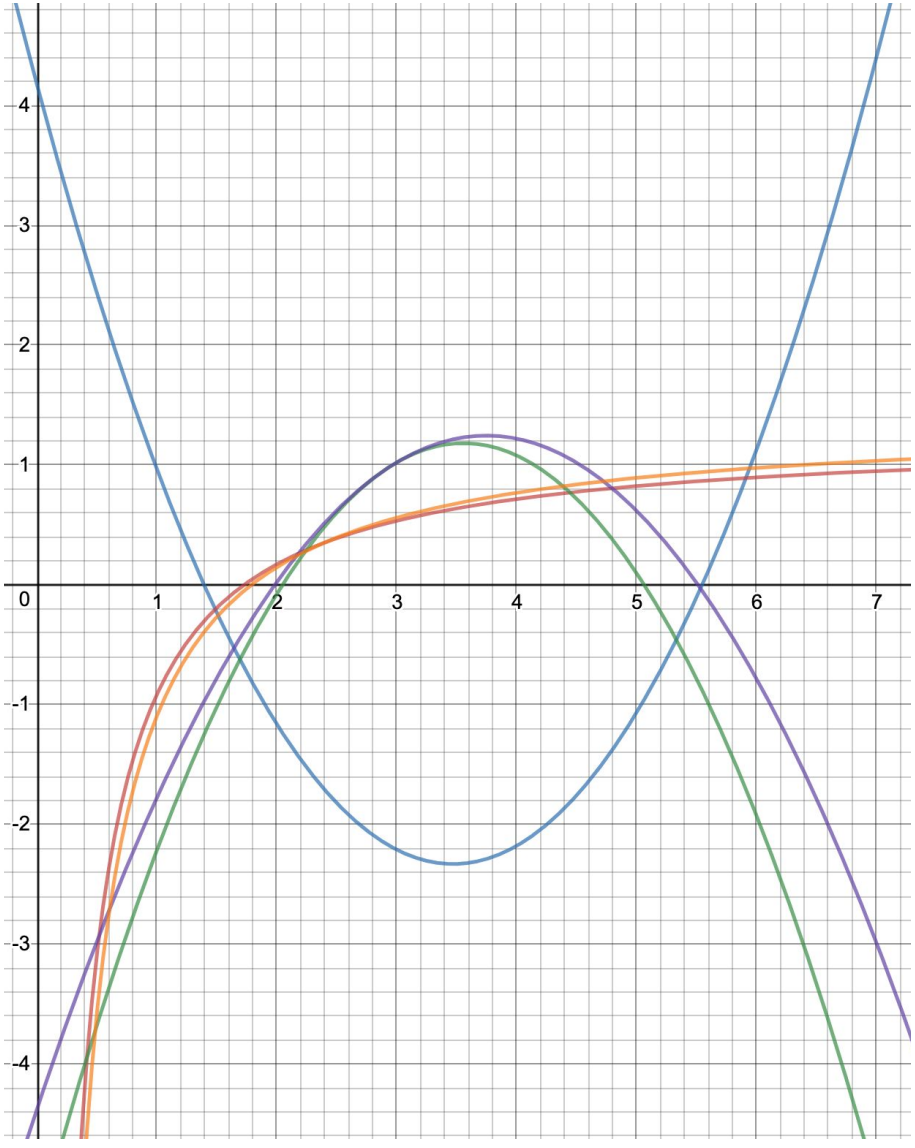


Values Over Time



This graph depicts a set of quadratic functions that nearly perfectly represent Renaissance influence on music over time. Each curve fits the data of one composer. The x-axis is time period (therefore x values outside of the range of [1, 5] are not representative of our data). The y-axis is the average correlation with that time period. The fact that parabolas that open upwards fit the data shows that Renaissance music was similar to general music during the Renaissance (of course), then slowly decreases in influence where it reaches its lowest around the Classical time period, then slowly gains influence again until the 20th century when it is almost equally influential as during the Renaissance (this is ironic because it make the 20th century the Renaissance of the Renaissance). The curve that seems to be unique from the rest belongs to Monteverdi, who is known to be a transitional composer between the Renaissance and Baroque eras. All of the time periods except for Renaissance had most of their composers best fit by inverse regressions (though there still were a few composers who followed quadratic functions well).

Values Over Time



This graph shows the curves of best fit for each of our time periods. Blue is Renaissance, red is Baroque, green is Classical, purple is Romantic, and yellow is 20th Century. The equations for each of the time periods fit the general trend of most of the composers during that era, but they don't fit perfectly. For example, Romantic and Classical composers were best fit by a inverse regression, whereas their curves as time periods are better represented as quadratics.

Conclusions

Part 1: Analysis Conclusions

Due to the amount of metrics that we measured from each song, we have reached many conclusions. The first and most important one that ties all of our measurement together is that the Renaissance time period is a clear outlier from the rest of classical music. This can be seen in their usage of time signatures, their average note value, and sequences of repeated pitch and note value. Within the Renaissance, an even further outlier is the composer Byrd, who essentially used only one pitch in each song, and in general used a much lower average pitch than any other composer.

We proved our initial hypothesis by finding the following statistical concepts in classical music:

- Benford's Law
- Zipf's Law
- The Pareto Principle
- Parallel Curves
- Unusual graphs
- Normal distribution
- Unique power laws

Part 2: Formula Conclusions

As can be seen by our heatmaps, the most important of all our conclusions is backed up by the outputs of our formula: the Renaissance time period was completely different from all other eras of classical music. This not only solidifies our conclusion about Renaissance music, but also strengthens the reliability of the formula.

We also showed that transitional composers, like Monteverdi and Beethoven, have both correlation with their first time period, and their second, validating both our data and our method of finding correlation. This also gives us predictive power in finding transitional composers, allowing us to find composers and songs that have similarity to two consecutive time periods.

We also found that the Renaissance follows a quadratic curve that shows that over time its music is becoming more correlated and relevant. If current trends hold, this means that as time progresses, classical music will begin sounding more and more like the Renaissance time period.

We can also see that both Classical and Romantic follow quadratics with negative coefficients (parabolas that open downwards), meaning their correlation over time is the opposite of Renaissance. As time progresses, they are now becoming less relevant.

Lastly, 20th Century and Baroque both were best represented as inverse regressions, meaning that they are slowly increasing in relevance, slower than Renaissance.

Future Steps

Our project has an enormous amount of potential for elaboration.

There are many more opportunities for data collection. Firstly, the number of metrics we used were not even close to the number of aspects that exist in classical music. Second, there are more than five time periods of classical music. We can subdivide the ones we have, and add more before and after our small snippet of musical history. Third, our project incorporated a mere 500 songs, out of the millions that have been written. Fourth, due to a lack of time we were unable to analyze multiple lines of music at once, therefore we had to only analyze melody lines. We can further develop our data collection scripts to solve this problem, giving opportunity for the analysis of chords and harmonies.

Another way our project can be continued is by further analyzing the data we collected. This can be done by writing analysis automation programs, or by continuing our work manually. Either way, there are many more discoveries to be made.

Though it may be the form of music people are most familiar with, European music is not the only kind of song. Music is a part of nearly every culture, and every one of them is likely to have statistical patterns. We could analyze a different type of music to see if it follows the same laws and patterns as Western music.

Future Steps

Also, our formula is essentially a new kind of statistical test that incorporated weighting. This means that it has almost infinite applications to the real world outside of music.

Another thing we would like to do is analyze not just old, classical music, but to modern music as well. There are so many genres of music, like jazz, hip hop and rap. We want to determine whether the patterns we found are exclusive to classical music, or found within all genres of music.

In order to collect this immense amount of data, we would like to outsource computing in order to maximize efficiency - perhaps to the musical computing lab at Stanford, which produced the Humdrum Toolkit (which was a large inspiration to this project). This would allow us to quickly collect all of the necessary data points.

Finally, we would also like to organize our data geographically, and see what trends are specific to each country. This would allow us to quantify culture through music, which would definitely yield fascinating results.

Future Steps

The scripts we wrote for musical analysis are innovative and unique, and would be helpful if given public access. We already have a GitHub page at:

bit.ly/beats-in-bytes-repo

GitHub repositories can be edited by anyone willing to contribute, and are available to anyone, meaning our code is open source.

However, GitHub may not be where people look when hoping to analyze music, so we are in the process of developing a website which will have a simple drag-and-drop interface where any kern file can be analyzed in seconds.

Another way of giving people easy easy access to our programs is by turning it into an application. We are already under development for an application for the Mac operating system that performs a similar task to the website.

Beats in Bytes is already a helpful tool, but it has the potential to be revolutionary when its analysis tools are further available and accessible to the public.