Introduction to Musical Corpus Studies

Release 0.0.1

Fabian C. Moss

Dec 21, 2020

CONTENT

Warning: This material is still (heavily) under construction and might change throughout the course!

You can help improving the course and let me know about any errors and inconsistencies that you find or suggest other ways of improving the course.

Welcome!

These pages present the content of the course "Introduction to Musical Corpus Studies" at the Institute of Musicology, given at University of Cologne in Fall 2020.

In the last two decades *Musical Corpus Studies* evolved from a niche discipline into a veritable research area. The growing availability of digital and digitized musical data as well as the application and development of modern methodologies from computer science, machine learning, and data science cast new light on old musicological questions and generate entirely novel approaches to empirical music research.

Moreover, the general methodological and epistemological approach of Musical Corpus Studies allows to transcend traditional intra-musicological boundaries between its sub-disciplintes (historical/systematic/ethnological/...) without sacrificing the respective specific viewpoints and perspectives.

This course offers a fundamental and practical introduction into these topics. It demonstrates, explores, and critically reflects central thematic areas and methods by means of a number of case studies. In the engagement with these topics the course also introduces elementary methods from natural language and music processing, as well as statistics, data analysis and visualization.

The course is aimed at students at the undergraduate level who have little or no empirical background and are curious about quantitative approaches to musicology.

CHAPTER

ONE

ORGANIZATION

1.1 About this course

This course aims at providing an example-based introduction to the rapidly developing field of Musical Corpus Studies (MCS). Introducing a field that relies equally on musicological domain knowledge and skills in computational and statistical methods faces obvious challenges: while most people interested in this field come with a background in either area, few people are versed in both, and it can take years to bridge the musicological-computational gap.

In particular, systematic introductions to programming or specific musicological topics can be at times quite arduous, even boring, because it takes a long time to proceed from learning basic concepts to acually interesting problems. The problems and "toy examples" that are presented to introduce the basic concepts are necessarily remote from real-world applications and challenging research problems.

This course takes an alternative route. It does not start with an introduction to the programming language Python (which will be used throughout to carry out the computational analyses) but rather showcases a number of recent corpus studies that take on musicological research questions. The focus thus lies in understanding how aspects of music can be studied with computational methods and by analyzing musical corpora.

If this sparks your interest, it will be much easier to pick up the basics for yourself, knowing what they are *for* and being motivated intrinsically. If you are not particularly interested in doing this kind of work yourself, you will still see a broad range of applications that are much more useful to you than learning (or not learning) programming basics.

1.2 Overview

This year's course takes place on two weekends (13-14 November and 11-12 December 2020), comprising twelve sessions in total. The topics cover a broad range of musicological topics, from folk melodies and Jazz solos, over harmonies in Beethoven's string quartets and 20th century Pop music, to Renaissance candences and metric patterns in Malian drum music (see Table 1.2).

No.	Date	Time	Topics
1	Fr., 13.11.2020	16:00-17:20 Uhr	Introduction / Background
2		17:40-19:00 Uhr	Melody I: Folk song melodies, notes, form
3	Sa., 14.11.2020	09:00-10:20 Uhr	Melody II: The melodic arc, intervals
4		10:40-12:00 Uhr	Melody III: Jazz solos
		12:00-13:00 Uhr	*Lunch Break*
5		13:00-14:20 Uhr	Group work
6		14:40-16:00 Uhr	Beethoven's string quartets
7	Fr., 11.12.2020	16:00-17:20 Uhr	Cadences in Renaissance polyphony (guest: Richard Freedman)
8		17:40-19:00 Uhr	
9	Sa., 12.12.2020	09:00-10:20 Uhr	Rhythm & Meter: Malian percussion music
10		10:40-12:00 Uhr	Timbre: Electronic Music 1950-1990
		12:00-13:00 Uhr	*Lunch Break*
11		13:00-14:20 Uhr	Group work
12		14:40-16:00 Uhr	Recapitulation and conclusion

1.3 Credits

Active participation in this course is compensated with 3 credit points (CPs), equivalent to a work load of 90 hours. These are distributed as follows: 24 SWS (à 45 minutes) are allocated to presence in the block seminar. Additionally, 24 SWS are dedicated to the preparation and follow-up of the material. The remainder of 42 SWS goes to the reading of the relevant literature.

1.4 Deliverables and learning objectives

Apart from attending and following the presentations by the lecturer, course work consists of three main parts: preparing the relevant literature (reading), completing the assigned exercises (group work), and critically engaging with the course materials in the form of a report written together with your group (report).

These deliverables will broaden your knowledge and understanding of current musicological research, enhance your organizational and social skills, and help you to develop efficient work-load management strategies. Finally, compiling a report will advance your communication and writing abilities.

Reading

For each session, the relevant literature is cited in the text and provided on ILIAS. Careful preparation of the reading material is required in order to be able to follow the content of the course. Because the course will mainly talk about methods and general points of musical corpus research, the content (and musical topic) will mainly be introduced by the literature.

I am aware that the reading workload is relatively high since the course will be taught as a block seminar and doesn't spread out over the entire semester. I hope that the fact that the course is finished before the end of the year compensates for this.

Group work

At the beginning of the course, you will be randomly assigned to a group. Together with your group (which will stay fixed for the entire semester), you will work on a number of exercises during the course, e.g. in Zoom breakout rooms. You will collaborate on specific tasks related to the topic at hand, discuss methodological questions, and help each other in the understanding of some of the concepts that are introduced in the course.

Report

After the course has ended, your group will be randomly assigned a course topic (one of the twelve sessions in Table 1.2). It is your task to write a report on this theme. The should be 6–8 pages long.

Questions that you could address are: What did you learn? Which concepts are not clear? Which methods did you (not) understand? What is missing? How can the textual descriptions be improved? Who in your group did what? Was the presentation of the material adequate? If not, what was missing? Please also write about the organization of your group, challenges and benefits.

Recommended structure for the report

1. **Introduction:** general description and summary of the course and your assigned session in particular.
2. **Discussion:** summarize the main discussion, open questions, and how you would continue this line or research.
3. **Issues:** describe in detail what was crucial for your understanding of the topic, what was missing, etc.
4. **Various:** anything that you would like to write in the report
5. **Author contributions:** describe briefly how each of you specifically contributed to the report.

Important: Submit your report by **31 January 2021, 23:59h** to fabian.moss@epfl.ch as a single PDF file per group, named *intro_corpusmus_<group_number>.pdf*, e.g. *intro_corpusmus_1.pdf*.

CHAPTER

TWO

INTRODUCTION AND BACKGROUND

Note: The slides for the introduction can be found here: pdf

CHAPTER

THREE

MELODIES IN FOLK SONGS

On Jupyter Hub, change the kernel to Python 3.7!

[1]:
```
import pandas as pd
import music21 as m21
import numpy as np
import statsmodels.api as sm

import matplotlib.pyplot as plt
import matplotlib as mpl

import seaborn as sns
sns.set_context("notebook")
```

[2]:
```
## Tragen Sie hier bitte Ihren username ein:
# USERNAME = "fmoss"

## for jupyter hubs
# %env QT_QPA_PLATFORM=offscreen
# # new user, create music21 environment variables.
# m21.environment.set('musicxmlPath', value='/usr/bin/mscore')
# m21.environment.set('musescoreDirectPNGPath', value='/usr/bin/mscore')
# m21.environment.set('graphicsPath', value=f'/home/{USERNAME}') # change accordingly␣
↪for your own username!
```

3.1 The *Essen Folksong Collection*

In this session, we work with a corpus of melodies, the *Essen Folksong Collection* (EFC). There are several ways to access this corpus, for example through the interface provided by the Center for Computer Assisted Research in the Humanities (CCARH) at Stanford University: http://essen.themefinder.org/ or via http://kern.ccarh.org/browse?l=essen.

A more convenient way to work with the pieces is by using the Python library `music21`. This library was developed and is maintaned my Mike Cuthbert at the MIT and is the most popular library for the computational analysis of symbolic music (i.e. scores). You can find its documentation here: http://web.mit.edu/music21/

However, using `music21` requires some training and getting used to its particular API (the way how to interact with its functions). We will not get into too many details here but rather showcase how it can be used for our purposes.

The first thing we do is to load the entire EFC and store it in a variable named `corpora`.

[3]:
```
# load corpus
corpora = m21.corpus.getComposer('essenFolksong')
```

Calling the variable `corpora` shows that it consists of a list of file paths. Using the `len()` function, we can find out how many corpora are stored in the variable `corpora`.

```
[4]: len(corpora)
```

```
[4]: 31
```

We can also directly call the variable `corpora` to see what it contains:

```
[5]: corpora
```

```
[5]: [WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/altdeu10.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/altdeu20.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad10.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad20.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad30.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad40.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad50.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad60.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad70.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/ballad80.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/boehme10.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/boehme20.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/dva0.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/erk10.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/erk20.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/erk30.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/erk5.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/fink0.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/folkHaydn.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/han1.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/han2.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/irl.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/kinder0.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/lot.abc'),
```

(continues on next page)

(continued from previous page)

```
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/lux.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/test0.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/test1.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/testd.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/teste.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/variant0.abc'),
 WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
↪essenFolksong/zuccal0.abc')]
```

The variable `corpora` is a list of file paths, each of which points to a corpus in this collection. Note that the location depends on the location where `music21` is installed. If you would do this on your own computer, you would see different paths. The file names at the end of the file paths indicate what they contain, e.g. `altdeu10.abc` contains old German folksongs, `boehme10.abc` contains Czech folksongs, and `han1.abc` contains Chinese folksongs.

The `.abc` file ending refers to the ABC notation for encoding melodies. You find more information about the ABC encoding here: http://abcnotation.com/

For example, a song could be encoded like this:

[6]:
```
example_song = """
X:1
T:Speed the Plough
M:4/4
C:Trad.
K:G
|:GABc dedB|dedB dedB|c2ec B2dB|c2A2 A2BA|
  GABc dedB|dedB dedB|c2ec B2dB|A2F2 G4:|
|:g2gf gdBd|g2f2 e2d2|c2ec B2dB|c2A2 A2df|
  g2gf g2Bd|g2f2 e2d2|c2ec B2dB|A2F2 G4:|
"""
```

The tripple quotes (`"""`) surrounding the ABC notation are used by Python to store multi-line text.

What can we already understand from this encoding?

`music21` can load this string and display a graphical output of the score. This is done by a **parser**. A parser is a program that reads a file and produces a structured output.

[7]:
```
parsed_example_song = m21.converter.parse(example_song)
```

We did not need to give it the entire string again because we have already saved it in the `example_song` variable. The purpose of variables is that you can refer to them later in your code without explicitly needing to state its value.

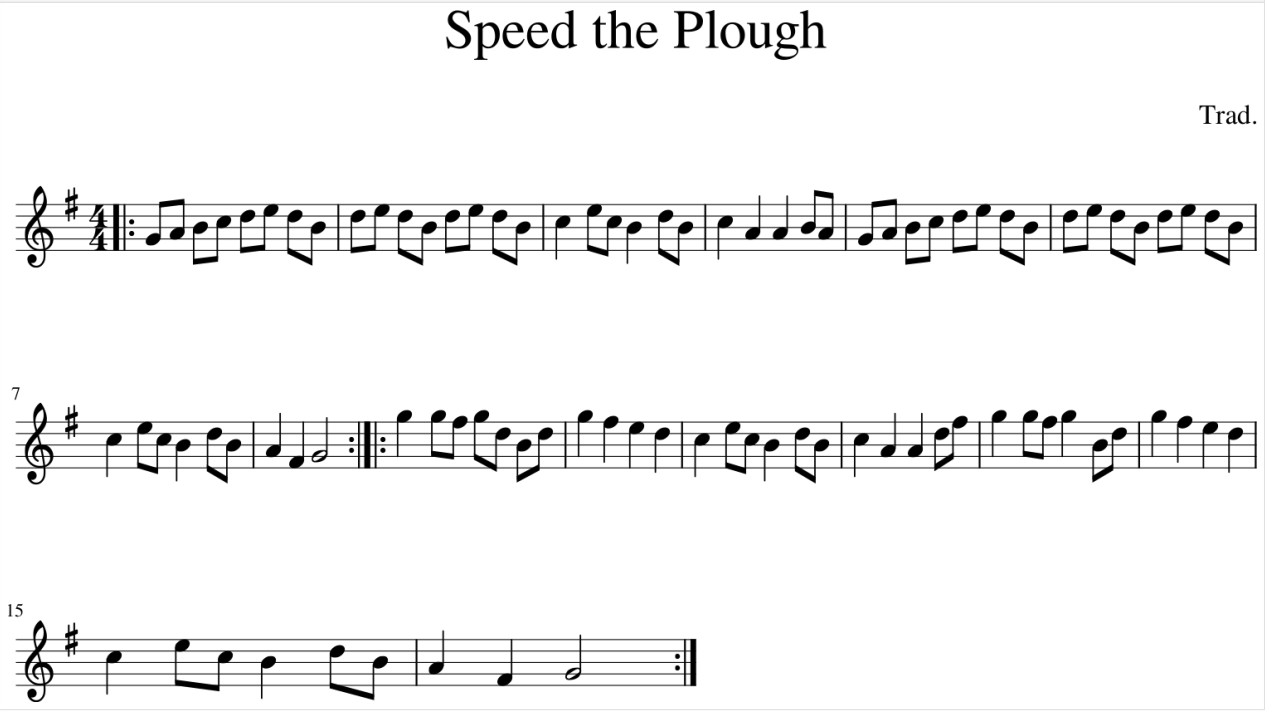
Calling the variable `parsed_example_song` now, however, does not really help us here...

[8]:
```
parsed_example_song
```

[8]:
```
<music21.stream.Score 0x1898a474340>
```

It returns a somewhat cryptic statement that says that the variable countains a `music21.stream.Score` object. Understanding the internal organization of `music21` goes beyond this class. For us, it is suffient to know that these objects have certain associated functions, called **methods**, that we can use on them. To look at the score of this example song, we use the method `.show()`.

[9]:
```
parsed_example_song.show()
```



Voilà, this is much better! Now, let us compare the score output to the ABC encoding of the song:

[10]:
```
print(example_song)
```

```
X:1
T:Speed the Plough
M:4/4
C:Trad.
K:G
|:GABc dedB|dedB dedB|c2ec B2dB|c2A2 A2BA|
  GABc dedB|dedB dedB|c2ec B2dB|A2F2 G4:|
|:g2gf gdBd|g2f2 e2d2|c2ec B2dB|c2A2 A2df|
  g2gf g2Bd|g2f2 e2d2|c2ec B2dB|A2F2 G4:|
```

Now the ABC notation makes already more sense. `T:Speed the Ploug` stands for the title, `M:4/4` for the meter, and `K:G` for the key of the song. The ABC documentation tells us that `X:1` encodes just a reference number, in case multiple pieces are stored in the same file (as in our case in the variable `corpora`, remember?). And the lines at the bottom encode the proper melody, where the letters represent note names that are organized into bars with or without repetition signs.

`music21` even gives us the option to listen to the song if we path the `midi` argument to the `.show()` method:

[11]:
```
parsed_example_song.show("midi")
```

```
<IPython.core.display.HTML object>
```

Now, what happens if we try to parse one of the corpora in the EFC? We can select a specific corpus by its **index** in the list. Python starts counting at 0, so the first file in the list corresponds to

```
[12]: corpora[0]
```

```
[12]: WindowsPath('C:/Users/fabianmoss/anaconda3/Lib/site-packages/music21/corpus/
      ↪essenFolksong/altdeu10.abc')
```

As you can see, this is just the first file path in the variable `corpora`. Let's try to parse it!

```
[13]: first_corpus = m21.converter.parse(corpora[0])
```

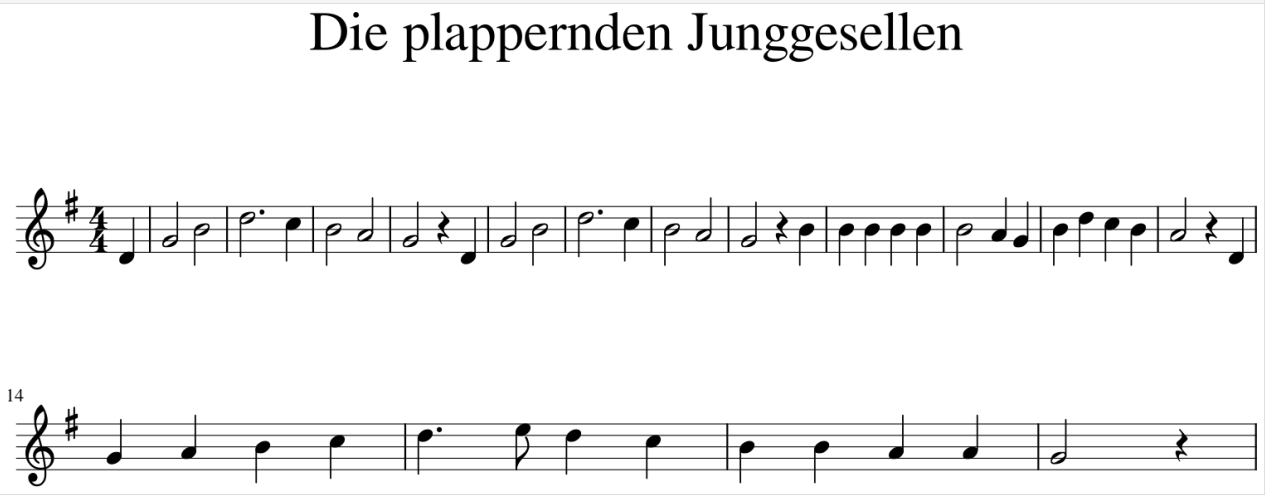
Looking at the new variable `first_corpus` shows a difference to the example song before; we don't have a `music21.stream.Score` object but a `music21.stream.Opus` object.

```
[14]: first_corpus
```

```
[14]: <music21.stream.Opus 0x1898b5ff4c0>
```

If we would call the `.show()` method on `first_corpus`, we would see the scores of all pieces that are in this particular corpus. But we don't know how many these are. It there are only three songs, it would not be a problem, but if there were thousands of songs, it could take a very long time to parse and display them all. Fortunately, all pieces in the collection have the `X:n` line that we saw above, so that we can directly reference them. With which number would we have to replace `n` if we wanted to look at the 7tst piece? Remember that Python starts counting at 0.

```
[15]: first_corpus[70].show()
```



A A B A'

```
[16]: first_corpus[70].show("midi")
```

```
<IPython.core.display.HTML object>
```

We have seen that we can select items from lists by **indexing** them, `list[i]`. We can get ranges of lists by using the `:` character. For example, `list[:10]` shows the first ten elements, `list[10:]` shows everything after the ninth element, and `list[3:6]` shows elements 3, 4, and 5 (not 6!) of the list.

3.2 Comparing songs

Looking at individual songs is interesting for music analysis but for that the computational approach is not really necessary. We could as easily do the same by just looking at a book of scores. The power of computational methods becomes clearer when we start comparing different songs, potentially in a large number.

To facilitate this comparison, we will first load all songs in all corpora of the EFC into a single list, called `songs` (this might take a couple of minutes).

[17]:
```
songs = [s for i in range(len(corpora)) for s in m21.converter.parse(corpora[i]) ]
```

This looks a bit complicated but all it does is to go through all corpora and extract all songs into a new list. The way we did it is called **list comprehension** in Python. It is not important if you don't understand this now but feel free to look it up!

Using the `len()` function again, we see how many songs we have in total.

[18]:
```
len(songs)
```

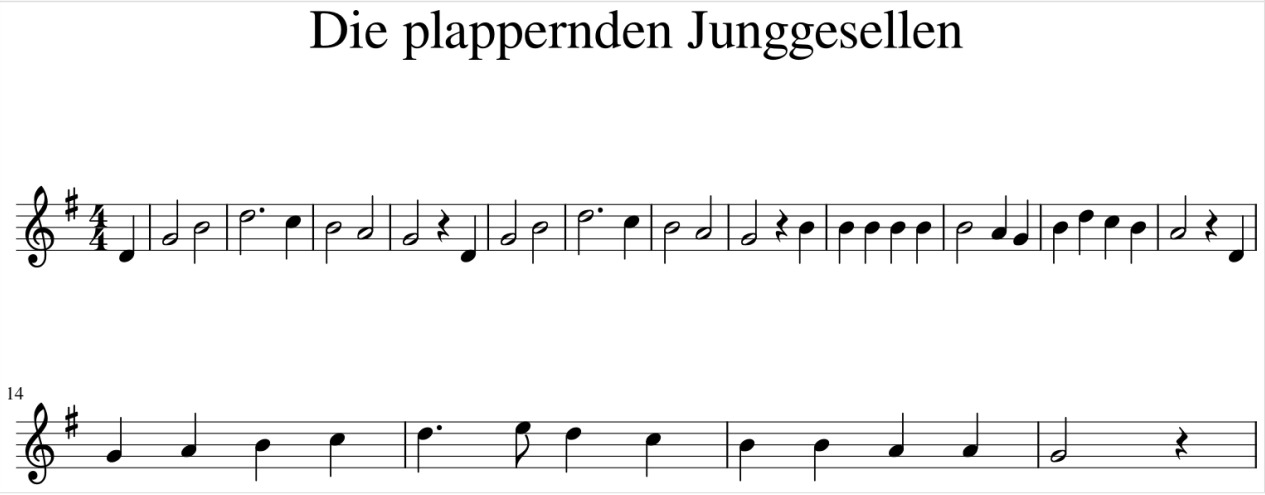
[18]:
```
8514
```

We can now use the list `songs` to compare two different songs. Again, we load the 71st song of the first corpus and store it now in a variable `german_song`, and we load chinese song with index 6200 into the variable `chinese_song`.

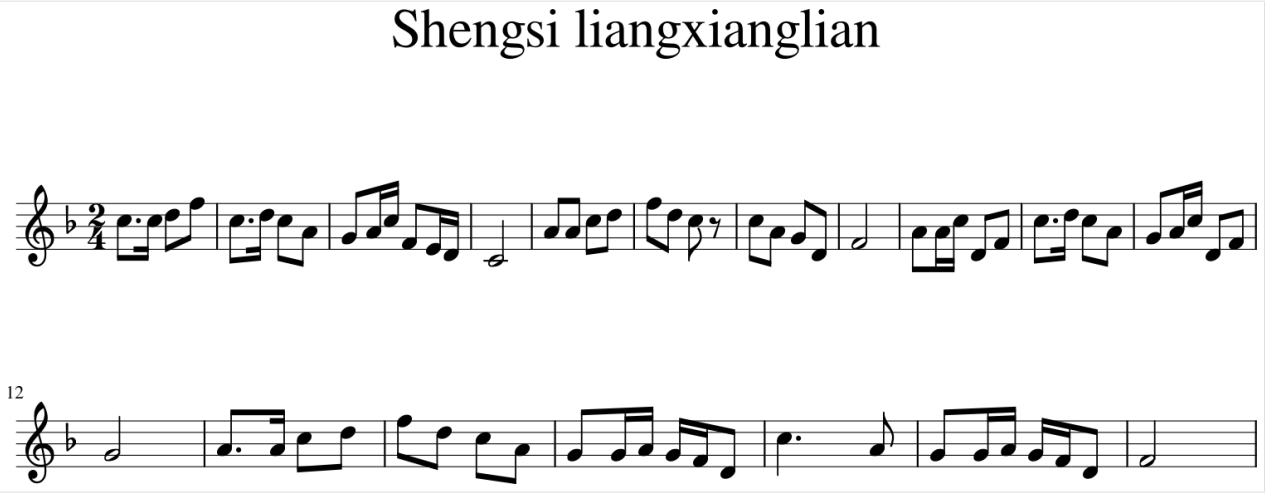
[19]:
```
german_song = songs[70]
chinese_song = songs[6200]
```

It is easy to display these songs now:

[20]:
```
german_song.show()
```



[21]:
```
chinese_song.show()
```



[22]:
```
chinese_song.show("midi")
```
```
<IPython.core.display.HTML object>
```

Analysis of songs...

3.3 Computational analysis

We now go on to a computational analysis of these two and all the other songs. Specifically, we wil compare their **melodic profiles**. To make things a bit simpler, we will just look at the notes.

A note can be easily represented as a pair of **pitch** (its height) and its **duration**. For example, the first note of the *Die plappernden Junggesellen* could be represented as `(D4, 1/4)`; it is a quarter note on the pitch D4 (the 4 indicates the octave in which the note is).

Another way to represent the pitch of notes is using **MIDI numbers**. MIDI stands for *Musical Instrument Digital Interface* and was developed for the communication between different electronic instruments such as keyboards. In MIDI, each note is simply associated with a number:

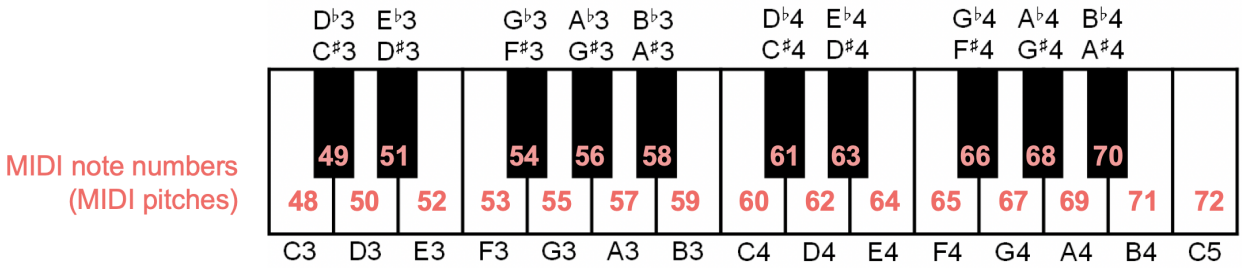


Image from https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S2_MIDI.html.

We can see that D4 is associated with the number 62. The second note, the G4, is associated with 62+5=67 because G is five semitones above D.

To make it easier to work with pieces in this way, we define a **function** that gives us a list of notes for each piece.

[23]:
```
def notelist(piece):
    """
    This function takes a song as input and returns a list of (pitch, duration) pairs,
    where the duration is given in quarter notes.
```

(continues on next page)

(continued from previous page)

```
    """

    df = pd.DataFrame([ (note.pitch.midi, note.quarterLength) for note in piece.flat.
↪notes ], columns=["MIDI Pitch", "Duration"])
    df["Onset"] = df["Duration"].cumsum()

    return df
```

Note that the duration of a note is given in quarter notes, i.e. a quarter note has a duration of 1, a half note has a duration of 2, and an eighth note has a duration of 0.5.

Let's display the first phrase (the first eight notes) of the German song:

[24]:
```
notelist(german_song)[:8]
```

[24]:

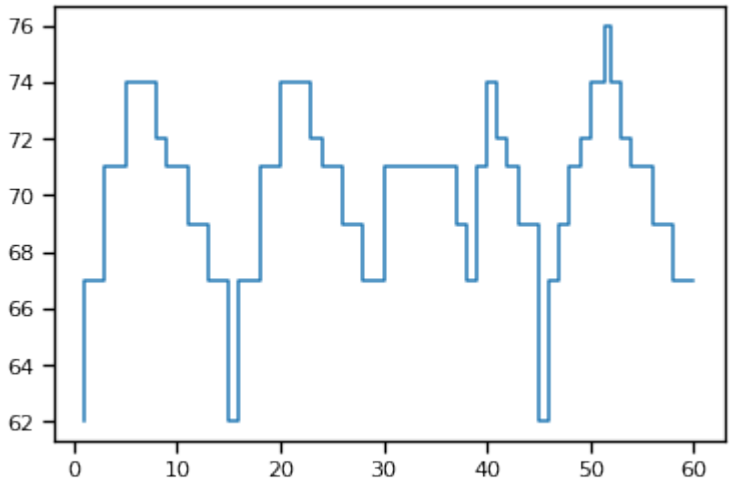
	MIDI Pitch	Duration	Onset
0	62	1.0	1.0
1	67	2.0	3.0
2	71	2.0	5.0
3	74	3.0	8.0
4	72	1.0	9.0
5	71	2.0	11.0
6	69	2.0	13.0
7	67	2.0	15.0

Note that we added another column, "Onset". What does it represent?

This allows us now to look at the **melodic profile** of a particular song.

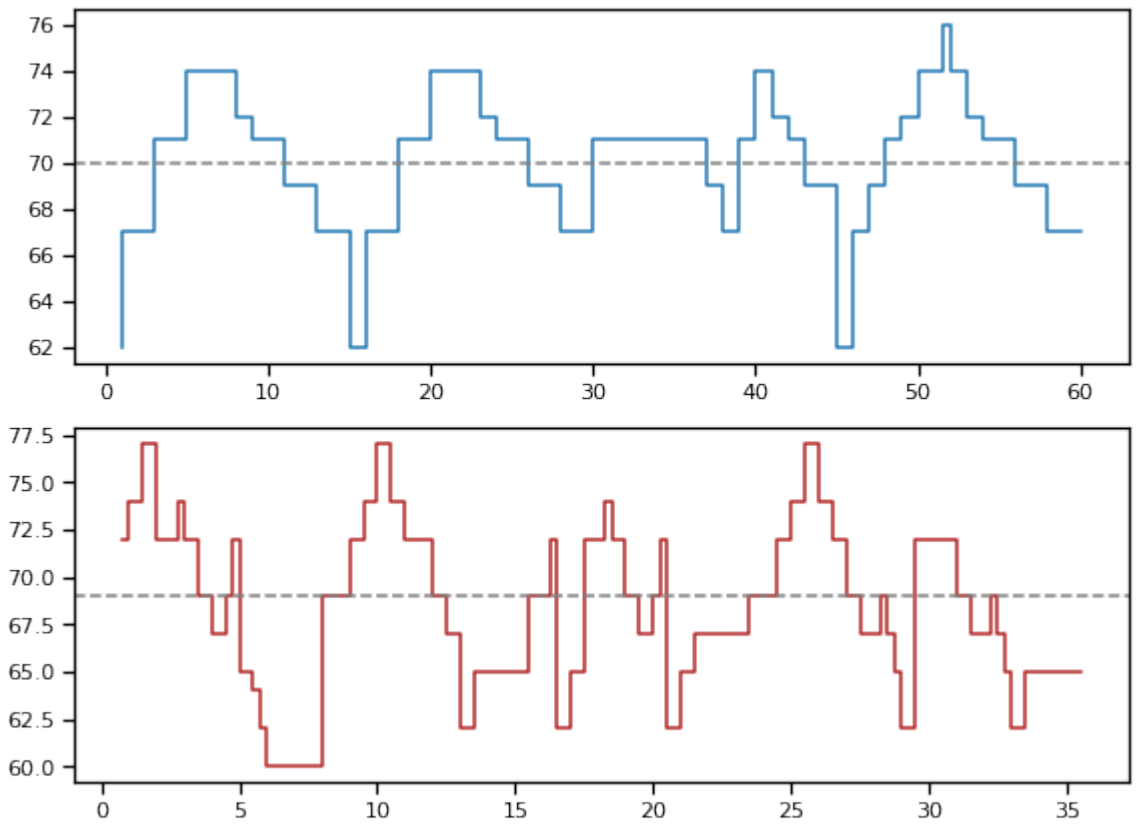
[25]:
```
def plot_melodic_profile(notelist, ax=None, c=None, mean=False, Z=False,␣
↪sections=False, standardized=False):

    if ax == None:
        ax = plt.gca()

    if standardized:
        x = notelist["Rel. Onset"]
        y = notelist["Rel. MIDI Pitch"]
    else:
        x = notelist["Onset"]
        y = notelist["MIDI Pitch"]

    ax.step(x,y, color=c)

    if mean:
        ax.axhline(y.mean(), color="gray", linestyle="--")

    if sections:
        for l in [ x.max() * i for i in [ 1/4, 1/2, 3/4] ]:
            ax.axvline(l, color="gray", linewidth=1, linestyle="--")
```

[26]:
```
plot_melodic_profile(notelist(german_song))
```



Likewise, we can as easily plot the melodic contour of the Chinese song (we will use a different color).

[27]:
```
fig, axes = plt.subplots(2,1, figsize=(8,6))

plot_melodic_profile(notelist(german_song), ax=axes[0], mean=True)
plot_melodic_profile(notelist(chinese_song), ax=axes[1], c="firebrick", mean=True)

plt.tight_layout()
plt.savefig("img/melodic_profiles.png")
```



The dashed grey lines in both plots show the average MIDI pitch of the song.

But still, it is quite difficult to compare them directly. They differ both with respect to their length (see the numbers on the "Onset" axis) and their pitches (see "MIDI Pitch" axis).

We need to transform them in a way that makes them directly comparable. To that end, we define a new function `standardize()`.

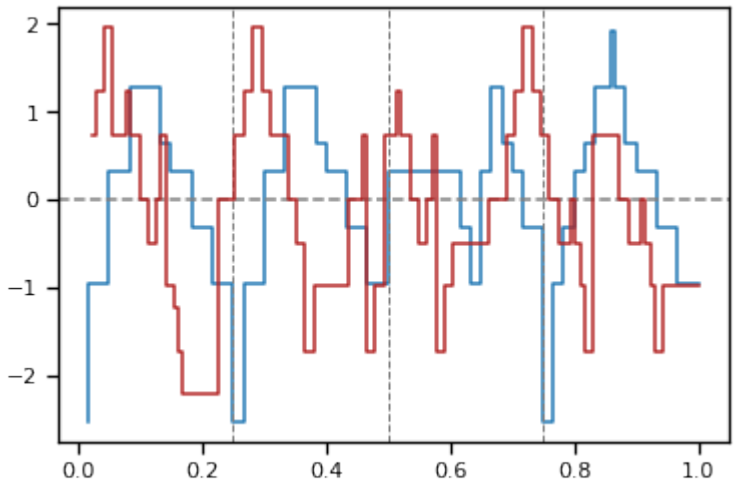
[28]:
```
def standardize(notelist):
    """
    Takes a notelist as input and returns a standardized version.
    """

    notelist["Rel. MIDI Pitch"] = (notelist["MIDI Pitch"] - notelist["MIDI Pitch"].
↪mean()) / notelist["MIDI Pitch"].std()
    notelist["Rel. Duration"] = notelist["Duration"] / notelist["Duration"].sum()
    notelist["Rel. Onset"] = notelist["Onset"] / notelist["Onset"].max()

    return notelist
```

[29]:
```
standardize(notelist(german_song))[:8]
```

[29]:

	MIDI Pitch	Duration	Onset	Rel. MIDI Pitch	Rel. Duration	Rel. Onset
0	62	1.0	1.0	-2.543827	0.016667	0.016667
1	67	2.0	3.0	-0.949300	0.033333	0.050000
2	71	2.0	5.0	0.326322	0.033333	0.083333
3	74	3.0	8.0	1.283038	0.050000	0.133333
4	72	1.0	9.0	0.645227	0.016667	0.150000
5	71	2.0	11.0	0.326322	0.033333	0.183333
6	69	2.0	13.0	-0.311489	0.033333	0.216667
7	67	2.0	15.0	-0.949300	0.033333	0.250000

[30]:
```
plot_melodic_profile(standardize(notelist(german_song)), mean=True, sections=True,␣
↪standardized=True)
plot_melodic_profile(standardize(notelist(chinese_song)), c="firebrick",␣
↪standardized=True)
```



Standardizing the songs makes it possible to compare them directly: They have now the same length 1 and their pitches are centered around the mean 0 with a standard deviation of 1.

However, already with two pieces this plot is quite crowded.

[31]:
```
dfs = [ ]

for i, song in enumerate(songs):
    df = standardize(notelist(song))
    df["Song ID"] = i
    dfs.append(df)

big_df = pd.concat(dfs).reset_index(drop=True)
```

[53]:
```
big_df
```

[53]:
```
        MIDI Pitch  Duration  Onset  Rel. MIDI Pitch  Rel. Duration  \
0               67      2.00   2.00        -1.819039       0.013158
1               70      2.00   4.00        -0.741977       0.013158
2               71      2.00   6.00        -0.382956       0.013158
3               72      2.00   8.00        -0.023935       0.013158
4               72      2.00  10.00        -0.023935       0.013158
...            ...       ...    ...              ...            ...
450591          71      0.25  28.50         0.691456       0.008197
450592          69      0.25  28.75         0.098779       0.008197
450593          73      0.25  29.00         1.284133       0.008197
450594          71      1.00  30.00         0.691456       0.032787
450595          69      0.50  30.50         0.098779       0.016393

        Rel. Onset  Song ID  Avg. MIDI Pitch  shifted_pitch
0         0.013158        0               72             -5
1         0.026316        0               72             -2
2         0.039474        0               72             -1
3         0.052632        0               72              0
4         0.065789        0               72              0
...            ...      ...              ...            ...
450591    0.934426     8513               68              3
450592    0.942623     8513               68              1
450593    0.950820     8513               68              5
450594    0.983607     8513               68              3
450595    1.000000     8513               68              1

[450596 rows x 9 columns]
```

[63]:
```
big_df.to_csv("data/big_df.csv") # comma-separated values
```

[66]:
```
big_df = pd.read_csv("data/big_df.csv")
```

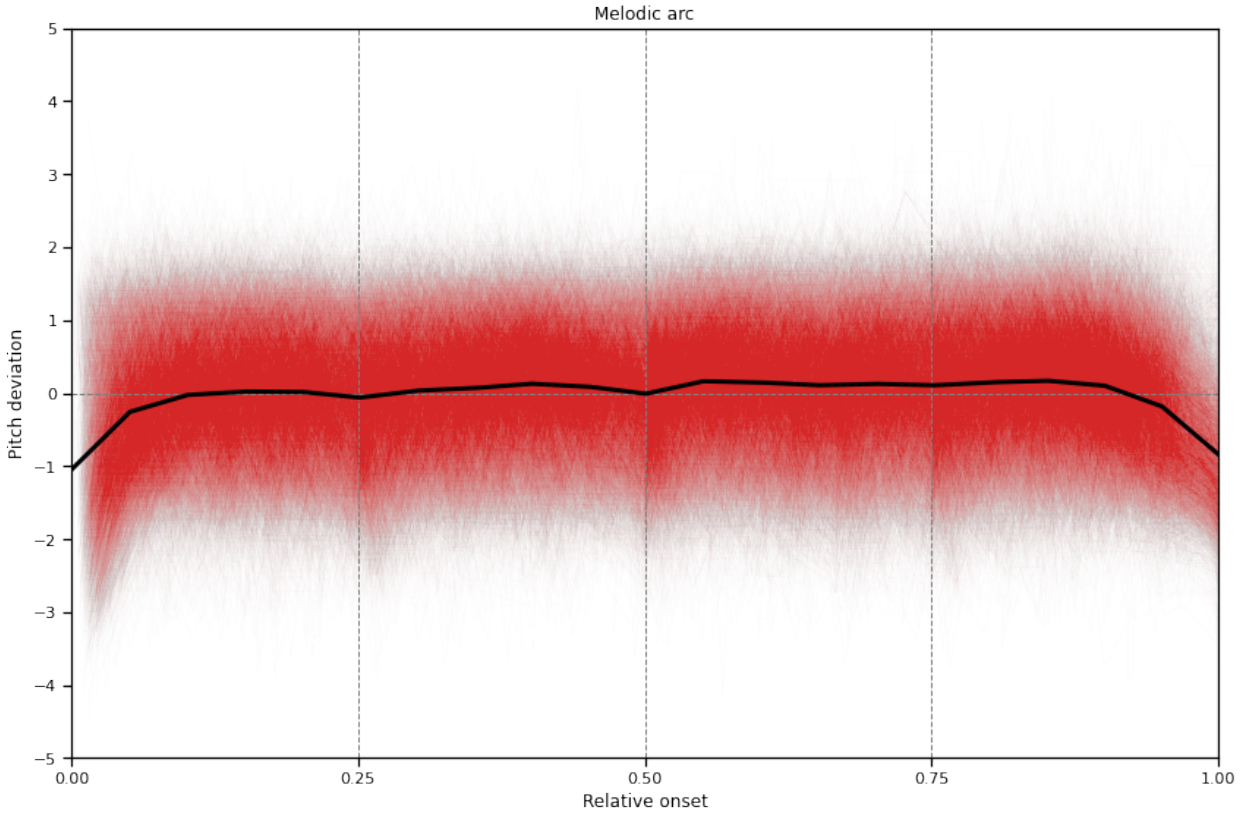
3.4 The melodic arc

[35]:
```
%%time

fig, ax = plt.subplots(figsize=(12,8))

grouped = big_df.groupby("Song ID")

for i, g in grouped:
    x = g["Rel. Onset"]
    y = g["Rel. MIDI Pitch"]
```

(continues on next page)

(continued from previous page)

```
    ax.plot(x,y, lw=.5, c="tab:red", alpha=1/100)

ax.axvline(.25, lw=1, ls="--", c="gray")
ax.axvline(.5, lw=1, ls="--", c="gray")
ax.axvline(.75, lw=1, ls="--", c="gray")
ax.axhline(0, lw=1, ls="--", c="gray")

lowess = sm.nonparametric.lowess
big_x = big_df["Rel. Onset"]
big_y = big_df["Rel. MIDI Pitch"]
big_z = lowess(big_y, big_x, frac=5/100, delta=1/20) # Locally-Weighted Scatterplot␣
↪Smoothing
ax.plot(big_z[:,0], big_z[:,1], c="black", lw=3)

plt.title("Melodic arc")
plt.xlabel("Relative onset")
plt.ylabel("Pitch deviation")
plt.xticks(np.linspace(0,1,5))
plt.yticks(np.linspace(-5,5,11))
plt.xlim(0,1)

plt.tight_layout()
plt.savefig("img/melodic_arc.png")
plt.show()
```



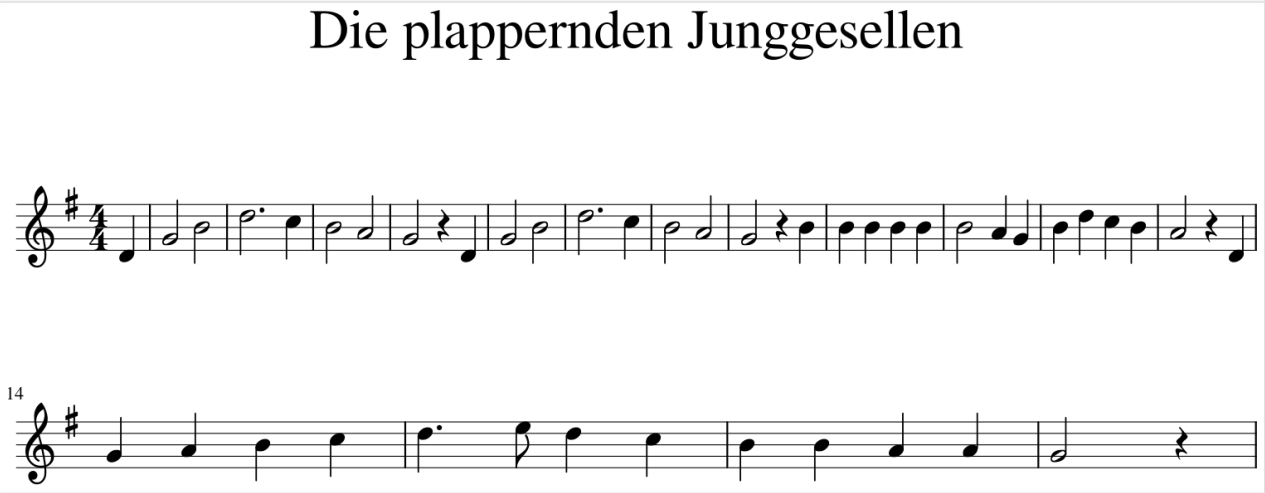

```
Wall time: 24.1 s
```

3.5 Intervals

We have seen that the melodic arc emerges as a stable shape over the entire EFC, and that sub-phrases of the songs likewise have an arc-like shape. In the remainder of this section, we look at **intervals**, the distance between two notes.

Let's come back to the song *Die plappernden Junggesellen*

[36]:
```
german_song.show()
```



We have already extracted its notes and stored them in a DataFrame:

[69]:
```
big_df[ big_df["Song ID"] == 70].head(8)
```

[69]:

	Unnamed: 0	Unnamed: 0.1	MIDI Pitch	Duration	Onset	Rel. MIDI Pitch	Rel. Duration	Rel. Onset	Song ID
2969	2969	2969	62	1.0	1.0	-2.543827	0.016667	0.016667	70
2970	2970	2970	67	2.0	3.0	-0.949300	0.033333	0.050000	70
2971	2971	2971	71	2.0	5.0	0.326322	0.033333	0.083333	70
2972	2972	2972	74	3.0	8.0	1.283038	0.050000	0.133333	70
2973	2973	2973	72	1.0	9.0	0.645227	0.016667	0.150000	70
2974	2974	2974	71	2.0	11.0	0.326322	0.033333	0.183333	70
2975	2975	2975	69	2.0	13.0	-0.311489	0.033333	0.216667	70
2976	2976	2976	67	2.0	15.0	-0.949300	0.033333	0.250000	70

The code above reads as "Select all rows in `big_df` for which the column `Song ID` is equal to 70". The `.head()` method displays the first 5 rows by default but you can specify the number of rows you want to be displayed (here 8).

Focusing on the "MIDI Pitch" column, the notes in the first phrase have MIDI pitch 62, 67, 71, 74, 72. Since intervals correspond to the difference between notes, the intervals for the beginning of this song are:

- +5 (67-62)
- +4 (71-67)

- +3 (74-71)
- -2 (72-74)
- -1 (71-72)
- -2 (69-71)
- -2 (67-69)

The sequence of intervals in this phrase is thus `[+5, +4, +3, -2, -1, -2, -2]`. The signs (+ or -) also reflect the arc-like shape of this first phrase, but the sizes of the intervals are not perfecly balanced. Note that `-2` (two descending semitones, or one descending whole tone) is the most frequent interval.

[38]:
```
all_ints = [ p2 - p1 for i, g in big_df.groupby("Song ID") for p1, p2 in zip(g["MIDI␣
↪Pitch"], g["MIDI Pitch"][1:]) ]
min_int = min(all_ints)
max_int = max(all_ints)
```

[39]:
```
min_int, max_int
```

[39]:
```
(-25, 25)
```

[40]:
```
len(all_ints)
```

[40]:
```
442082
```

[41]:
```
ints_df = pd.DataFrame(0, index=np.arange(min_int,max_int), columns=np.arange(min_int,
↪max_int+1))

for i, g in big_df.groupby("Song ID"):
    intervals = [ p2 - p1 for p1, p2 in zip(g["MIDI Pitch"], g["MIDI Pitch"][1:])]

    for i1, i2 in zip(intervals, intervals[1:]):
        ints_df.loc[i1,i2] += 1
```

[73]:
```
ints_df.loc[-10:10, -10:10]
```

[73]:

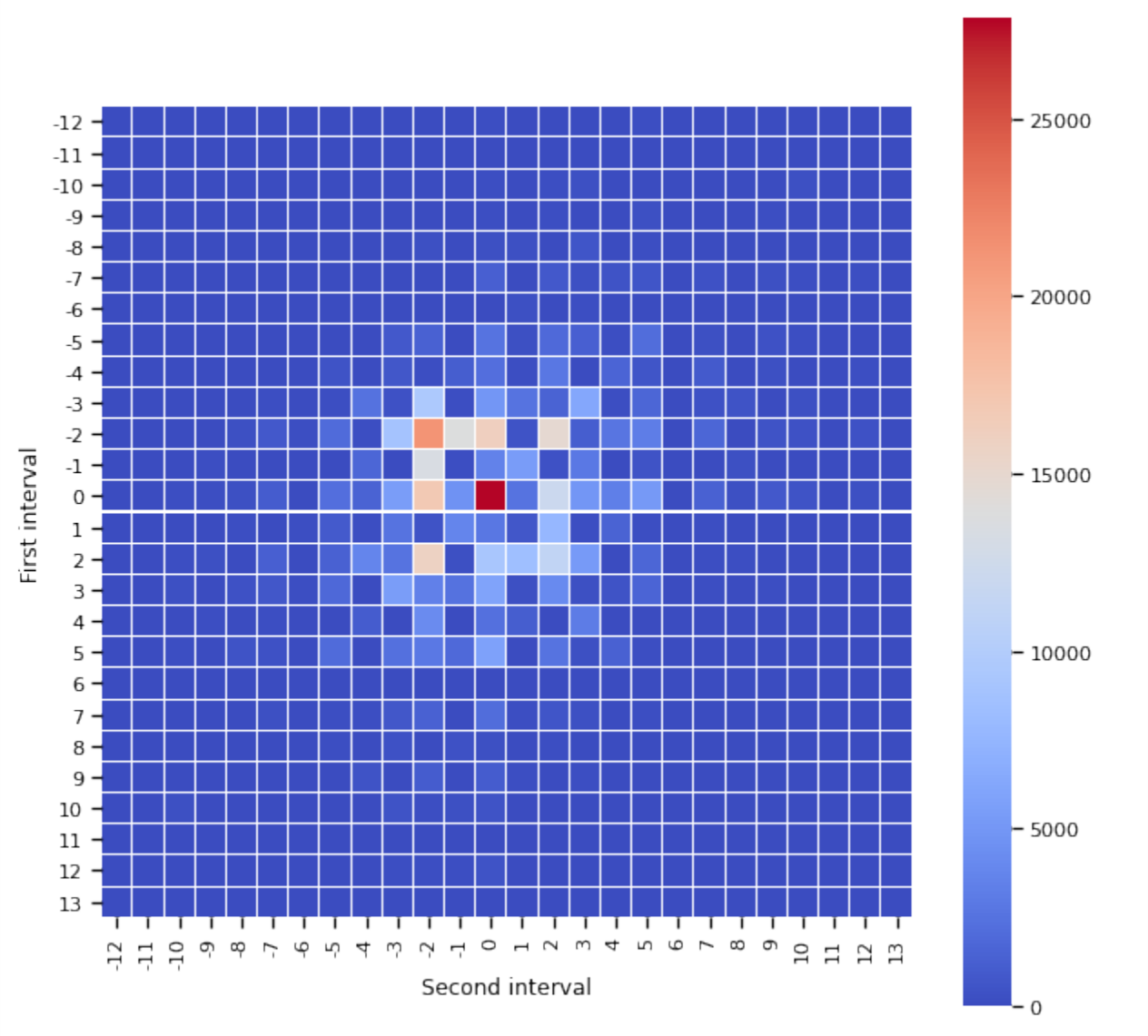
	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	...	1	\
-10	0	0	0	0	0	2	3	2	34	0	...	16	
-9	0	0	0	6	0	3	0	39	10	31	...	5	
-8	0	1	0	1	0	0	19	0	319	5	...	302	
-7	0	0	2	14	0	37	2	91	96	103	...	17	
-6	0	0	0	0	0	0	12	11	8	21	...	274	
-5	1	0	1	49	0	75	32	866	1361	25	...	230	
-4	3	0	27	11	5	461	18	692	167	1099	...	129	
-3	1	91	0	192	2	215	2490	416	9478	134	...	2547	
-2	67	14	260	858	132	1964	113	8871	21285	13896	...	454	
-1	5	174	4	68	47	32	1679	91	13445	205	...	5410	
0	186	130	310	1022	80	2270	1440	5506	16887	4603	...	2578	
1	55	6	174	43	110	944	165	2564	501	3765	...	747	
2	138	294	29	1288	15	1361	3754	2562	15795	254	...	8404	
3	272	23	373	655	122	1755	24	5509	3397	2400	...	295	
4	5	164	3	130	11	51	1026	64	4217	60	...	1133	
5	116	23	505	330	13	1996	82	2375	2834	1868	...	102	
6	2	4	1	4	23	1	14	18	44	34	...	42	
7	31	18	11	273	3	167	128	665	1338	59	...	119	
8	47	1	88	7	19	149	4	370	33	384	...	23	
9	1	61	1	20	3	20	442	18	1024	4	...	122	

(continues on next page)

(continued from previous page)

```
 10  267    0   56    23   28    58     0   517    163    295  ...      3

         2     3     4     5    6     7    8    9    10
-10    174   219    60    91    0    78   67   29  243
-9     159    39    40   313    2    30    0  111   27
-8      66   605     3   159    2     4   73    0   32
-7     859   300   527   651    1   399   18  219  141
-6       7   132    25    10   19     6    3    3    4
-5    1906  1272   148  2131   22   252   87  368  160
-4    2738    40  1610   616   13   884    8  269   31
-3    1467  6274   109  1684   13   403  508   59  261
-2   14883  1115  2616  3216   63  1681   88  537  407
-1     396  2878    58   477   44    23  241    3   52
 0   12142  4947  3340  5203   30  1353  292  824  478
 1    7649   193  1470   132   26   172    9   59    2
 2   11261  5249    86  1695    2   187  171   57   61
 3    4128   231   519  1523   35   180    0  122   11
 4      67  3153    20   102    7     1   14    0    2
 5    2557   270  1355   160    1    94    0   16    5
 6       8    20     0     0    0     0    0    0    0
 7     566   249    12   166    0     8    2    6    0
 8     159     0    23     6    0     1    0    0    0
 9      13   129     1     8    0     0    5    0    0
 10     78     0     4     5    0     0    0    0    0

[21 rows x 21 columns]
```

[74]:
```
fig, ax = plt.subplots(figsize=(10,10))
sns.heatmap(ints_df.loc[-12:13,-12:13], cmap="coolwarm", square=True, linewidths=0.01,
↪ax=ax)
plt.ylabel("First interval")
plt.xlabel("Second interval")
plt.show()
```



The two most common interval pairs are `(0,0)` and `(-2,-2)`. A much less frequent pair of intervals is `(5,0)`, but this is still much more frequent than, for example, `(9,9)`.

To which melodic fragments do these correspond?

[44]:
```
big_df["Avg. MIDI Pitch"] = 0

for i, group in big_df.groupby("Song ID"):
    grp_mean_pitch = int(group["MIDI Pitch"].mean())
    big_df.loc[big_df["Song ID"] == i, "Avg. MIDI Pitch"] = grp_mean_pitch
```

[45]:
```
big_df["shifted_pitch"] = big_df["MIDI Pitch"] - big_df["Avg. MIDI Pitch"]
```

[46]:
```
big_df.tail()
```

[46]:

	MIDI Pitch	Duration	Onset	Rel. MIDI Pitch	Rel. Duration	\
450591	71	0.25	28.50	0.691456	0.008197	
450592	69	0.25	28.75	0.098779	0.008197	

(continues on next page)

(continued from previous page)

```
450593          73      0.25  29.00          1.284133       0.008197
450594          71      1.00  30.00          0.691456       0.032787
450595          69      0.50  30.50          0.098779       0.016393

        Rel. Onset  Song ID  Avg. MIDI Pitch  shifted_pitch
450591    0.934426     8513               68              3
450592    0.942623     8513               68              1
450593    0.950820     8513               68              5
450594    0.983607     8513               68              3
450595    1.000000     8513               68              1
```

[47]:
```
idx = np.arange(big_df["shifted_pitch"].min(), big_df["shifted_pitch"].max() + 1)
idx
```

[47]:
```
array([-16, -15, -14, -13, -12, -11, -10,  -9,  -8,  -7,  -6,  -5,  -4,
        -3,  -2,  -1,   0,   1,   2,   3,   4,   5,   6,   7,   8,   9,
        10,  11,  12,  13,  14,  15,  16,  17])
```

[48]:
```
transitions_df = pd.DataFrame(0, index=idx, columns=idx)
transitions_df
```

[48]:

	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	...	8	9	10
-16	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-15	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-14	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-13	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-12	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-11	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-10	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-9	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-8	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-7	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-6	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-5	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-4	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
-1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0
10	0	0	0	0	0	0	0	0	0	0	...	0	0	0
11	0	0	0	0	0	0	0	0	0	0	...	0	0	0
12	0	0	0	0	0	0	0	0	0	0	...	0	0	0
13	0	0	0	0	0	0	0	0	0	0	...	0	0	0
14	0	0	0	0	0	0	0	0	0	0	...	0	0	0
15	0	0	0	0	0	0	0	0	0	0	...	0	0	0
16	0	0	0	0	0	0	0	0	0	0	...	0	0	0
17	0	0	0	0	0	0	0	0	0	0	...	0	0	0

(continues on next page)

(continued from previous page)

```
      11   12   13   14   15   16   17
-16    0    0    0    0    0    0    0
-15    0    0    0    0    0    0    0
-14    0    0    0    0    0    0    0
-13    0    0    0    0    0    0    0
-12    0    0    0    0    0    0    0
-11    0    0    0    0    0    0    0
-10    0    0    0    0    0    0    0
-9     0    0    0    0    0    0    0
-8     0    0    0    0    0    0    0
-7     0    0    0    0    0    0    0
-6     0    0    0    0    0    0    0
-5     0    0    0    0    0    0    0
-4     0    0    0    0    0    0    0
-3     0    0    0    0    0    0    0
-2     0    0    0    0    0    0    0
-1     0    0    0    0    0    0    0
 0     0    0    0    0    0    0    0
 1     0    0    0    0    0    0    0
 2     0    0    0    0    0    0    0
 3     0    0    0    0    0    0    0
 4     0    0    0    0    0    0    0
 5     0    0    0    0    0    0    0
 6     0    0    0    0    0    0    0
 7     0    0    0    0    0    0    0
 8     0    0    0    0    0    0    0
 9     0    0    0    0    0    0    0
 10    0    0    0    0    0    0    0
 11    0    0    0    0    0    0    0
 12    0    0    0    0    0    0    0
 13    0    0    0    0    0    0    0
 14    0    0    0    0    0    0    0
 15    0    0    0    0    0    0    0
 16    0    0    0    0    0    0    0
 17    0    0    0    0    0    0    0

[34 rows x 34 columns]
```

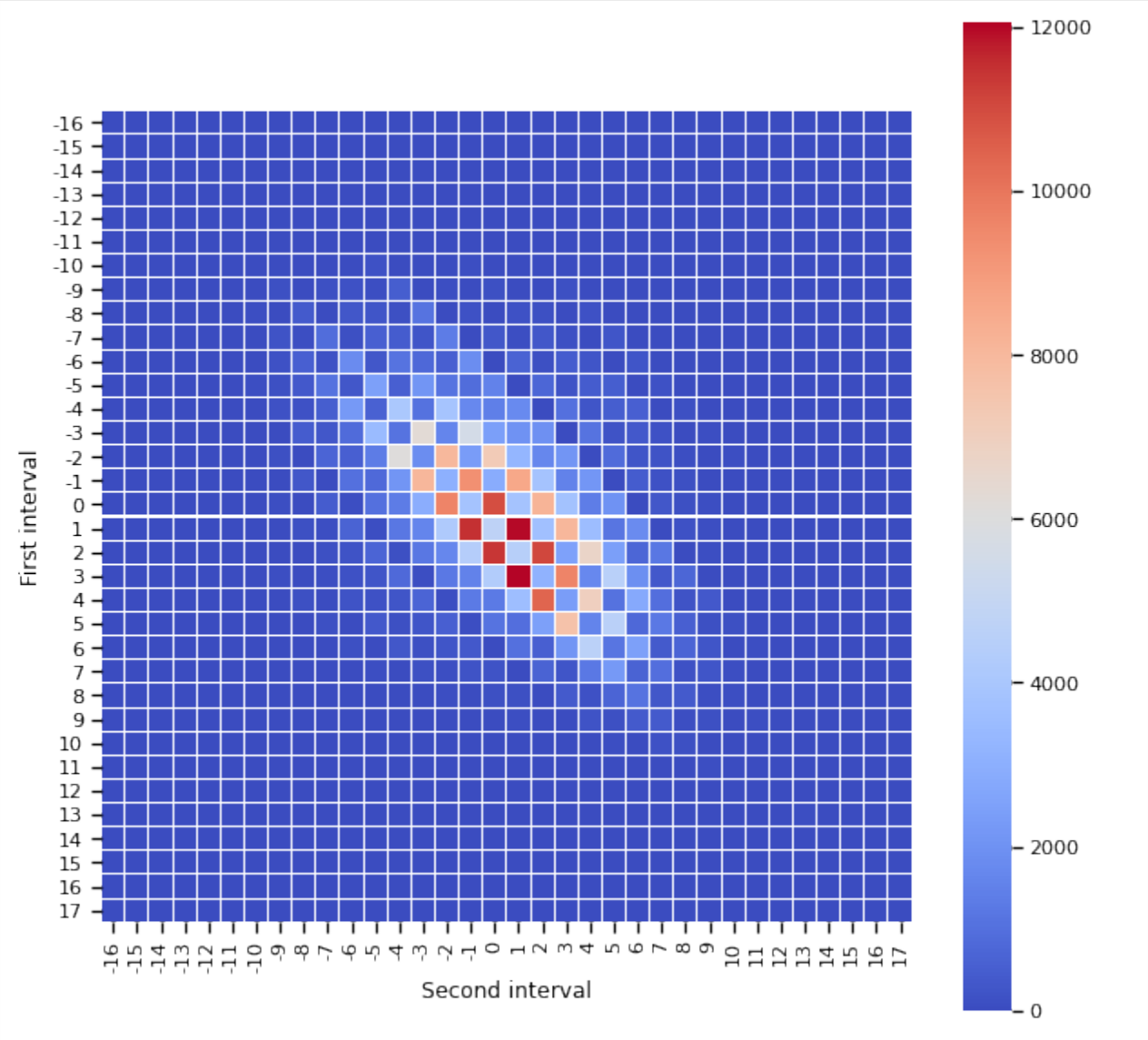
[49]:
```
%%time

for i, group in big_df.groupby("Song ID"):
    for bg in zip(group["shifted_pitch"], group["shifted_pitch"][1:]):
        transitions_df.loc[bg[0],bg[1]] +=1
```

```
Wall time: 1min 29s
```

[50]:
```
print(f"There are {transitions_df.sum().sum()} intervals in total in the corpus.")
```

```
There are 442082 intervals in total in the corpus.
```

[51]:
```
fig, ax = plt.subplots(figsize=(10,10))

g = sns.heatmap(transitions_df, cmap="coolwarm", linewidths=.01, square=True)
plt.ylabel("First interval")
plt.xlabel("Second interval")
plt.show()
```


First interval
Second interval
-16
-15
-14
-13
-12
-11
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
12000
10000
8000
6000
4000
2000
0

[ ]:

CHAPTER

FOUR

SOLOS IN THE *WEIMAR JAZZ DATABASE*

Disclaimer: I am not the expert here!

In this session, we will have a look at is the Jazzomat Research Project that contains the *Weimar Jazz Database* (WJazzD). Let us first browse the site.

One of the outcomes of this research project is the freely-available book:

- Pfleiderer, M., Frieler, K., Abeßer, J., Zaddach, W.-G., & Burkhard, B. (Eds.) (2017). Inside the Jazzomat. New Perspectives for Jazz Research. Mainz: Schott Campus (Open Access).

[1]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import sqlite3

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("white")
sns.set_context("talk")
```

The WJazzD can be downloaded at https://jazzomat.hfm-weimar.de/download/download.html A local copy of the database is stored at `data/wjazz.db`. We use the `sqlite3` library to connect to this database.

[2]:
```
conn = sqlite3.connect("data/wjazzd.db")
```

[3]:
```
conn
```

[3]:
```
<sqlite3.Connection at 0x201747b7990>
```

We can now use `pandas` to read the data out of the database.

[4]:
```
solos = pd.read_sql("SELECT * FROM melody", con=conn)
```

The `"SELECT * FROM melody"` means "Select everything from the table 'melody' in the database". Let's look at the first ten entries.

Likewise, we can select the `composition_info` table that contains a lot of metadata for the solos:

[5]:
```
solos_meta = pd.read_sql("SELECT * from solo_info", con=conn)
```

The `.shape` attribute shows us how many solos are in the database.

[6]:
```
solos_meta.shape
```

[6]:
```
(456, 17)
```

The `.sample()` method draws a number of rows at random from a DataFrame.

[13]:
```
solos_meta.sample(5)
```

[13]:
```
     melid  trackid  compid  recordid      performer        title titleaddon  \
429    430      260     236       140  Wayne Shorter   Eighty-One
440    441      335     295       180     Woody Shaw     Rosewood
240    241      152     137        76  Johnny Hodges        Bunny
326    327      174     158        87    Miles Davis      Walkin'
224    225      199     177       103  John Coltrane  Impressions       1963

     solopart instrument    style  avgtempo tempoclass  rhythmfeel    key  \
429         1         ts  POSTBOP     138.4     MEDIUM       SWING  F-maj
440         1         tp  POSTBOP     171.7  MEDIUM UP  LATIN/FUNK
240         1         as    SWING     114.4     MEDIUM       SWING  C-maj
326         1         tp  HARDBOP     127.7     MEDIUM       SWING  F-maj
224         1         ts  POSTBOP     278.7         UP       SWING  D-min

    signature                                        chord_changes  chorus_count
429       4/4  A1: ||Fsus7   |Bbsus7   |Fsus7   |Fsus7   |Bbs...             5
440       4/4  A1: ||G-79 F-79 |G-79 F-79 |C-79 Bb-79 |Gb79  ...             1
240       4/4  A1: ||A-7 D7 |G E7 |A-7 D7 |D-7 G7 |C F7 |Bb B...             1
326       4/4  A1: ||F7   |Bb7   |F7   |F7   |Bb7   |Bb7   |F...             7
224       4/4  A1: ||D-7   |D-7   |D-7   |D-7   |D-7   |D-7  ...            13
```

The first rows of a DataFrame can be accessed with the `.head()` method...

[19]:
```
solos.head()
```

[19]:
```
   eventid  melid      onset  pitch  duration  period  division  bar  beat  \
0        1      1  10.343492   65.0  0.138776       4         1    0     1
1        2      1  10.637642   63.0  0.171247       4         4    0     2
2        3      1  10.843719   58.0  0.081270       4         4    0     2
3        4      1  10.948209   61.0  0.235102       4         1    0     3
4        5      1  11.232653   63.0  0.130612       4         1    0     4

   tatum  ...  f0_mod  loud_max   loud_med   loud_sd  loud_relpos loud_cent  \
0      1  ...          0.126209  66.526087  5.541147     0.307692  0.389466
1      1  ...          0.349751  69.133321  2.912412     0.250000  0.468687
2      4  ...          0.094051  66.352130  3.564563     0.428571  0.531354
3      1  ...          0.521187  66.484173  2.414298     0.818182  0.559333
4      1  ...          0.560737  71.699054  2.185794     0.166667  0.438973

   loud_s2b   f0_range  f0_freq_hz  f0_med_dev
0  1.056169  37.794261   12.932532   -0.328442
1  1.120317   6.365930    6.956935   11.135423
2  1.310389  68.010392         NaN   32.366787
3  0.984047  15.443906    5.867151   -3.374696
4  1.061262  11.444363    8.329975    6.377737

[5 rows x 26 columns]
```

... and the last rows with the `.tail()` method.

[16]:
```
solos.tail()
```

[16]:
```
        eventid  melid      onset  pitch  duration  period  division  bar  \
200804   200805    456  63.135057   57.0  0.168345       4         2   53
200805   200806    456  63.303401   55.0  0.087075       4         3   54
```

(continues on next page)

(continued from previous page)

```
200806   200807    456  63.390476   57.0  0.191565       4         3   54
200807   200808    456  63.640091   59.0  0.406349       4         1   54
200808   200809    456  64.058050   52.0  1.433832       4         2   54

        beat  tatum  ...    f0_mod  loud_max   loud_med   loud_sd  loud_relpos  \
200804     4      2  ...           1.113380  72.169552  6.896394     0.687500
200805     1      1  ...           0.491496  69.732265  1.814723     0.500000
200806     1      2  ...     slide 1.187058  76.628621  2.628726     0.411765
200807     2      1  ...           0.972676  66.042058  3.690577     0.000000
200808     3      2  ...   vibrato 0.368321  58.174931  9.418678     0.053030

       loud_cent  loud_s2b    f0_range  f0_freq_hz  f0_med_dev
200804  0.581956  1.271747  191.074095   10.966972  -11.891698
200805  0.595212  1.339060   40.375449         NaN  -99.173779
200806  0.590950  1.432802  104.823845   11.148561   -2.911604
200807  0.334937  1.082549  165.810976    2.659723   14.311001
200808  0.400571  1.278890   66.932198    2.153916   -9.381310

[5 rows x 26 columns]
```

As we already know, the `.shape` attribute shows the overall size of the table.

[20]:
```
solos.shape
```

[20]:
```
(200809, 26)
```

The `solos` table contains 26 columns that cannot be displayed at once. We can have a look at the column names by using the `.columns` attribute.

[21]:
```
solos.columns
```

[21]:
```
Index(['eventid', 'melid', 'onset', 'pitch', 'duration', 'period', 'division',
       'bar', 'beat', 'tatum', 'subtatum', 'num', 'denom', 'beatprops',
       'beatdur', 'tatumprops', 'f0_mod', 'loud_max', 'loud_med', 'loud_sd',
       'loud_relpos', 'loud_cent', 'loud_s2b', 'f0_range', 'f0_freq_hz',
       'f0_med_dev'],
      dtype='object')
```

A description of what these columns contain is stated on the website: https://jazzomat.hfm-weimar.de/dbformat/dbformat.html

For our analyses it will be usefull to have also the name of the performer in the `solos` DataFrame. We create a **dictionary** that maps the `melid` (unique identification number for each solo) to the name of the performer.

[26]:
```
mapper = dict( solos_meta[["melid", "performer"]].values )
mapper
```

[26]:
```
{1: 'Art Pepper',
 2: 'Art Pepper',
 3: 'Art Pepper',
 4: 'Art Pepper',
 5: 'Art Pepper',
 6: 'Art Pepper',
 7: 'Benny Carter',
 8: 'Benny Carter',
 9: 'Benny Carter',
 10: 'Benny Carter',
 11: 'Benny Carter',
```

(continues on next page)

(continued from previous page)

```
12: 'Benny Carter',
13: 'Benny Carter',
14: 'Benny Goodman',
15: 'Benny Goodman',
16: 'Benny Goodman',
17: 'Benny Goodman',
18: 'Benny Goodman',
19: 'Benny Goodman',
20: 'Benny Goodman',
21: 'Ben Webster',
22: 'Ben Webster',
23: 'Ben Webster',
24: 'Ben Webster',
25: 'Ben Webster',
26: 'Bix Beiderbecke',
27: 'Bix Beiderbecke',
28: 'Bix Beiderbecke',
29: 'Bix Beiderbecke',
30: 'Bix Beiderbecke',
31: 'Bob Berg',
32: 'Bob Berg',
33: 'Bob Berg',
34: 'Bob Berg',
35: 'Bob Berg',
36: 'Bob Berg',
37: 'Bob Berg',
38: 'Branford Marsalis',
39: 'Branford Marsalis',
40: 'Branford Marsalis',
41: 'Branford Marsalis',
42: 'Branford Marsalis',
43: 'Branford Marsalis',
44: 'Buck Clayton',
45: 'Buck Clayton',
46: 'Buck Clayton',
47: 'Cannonball Adderley',
48: 'Cannonball Adderley',
49: 'Cannonball Adderley',
50: 'Cannonball Adderley',
51: 'Cannonball Adderley',
52: 'Charlie Parker',
53: 'Charlie Parker',
54: 'Charlie Parker',
55: 'Charlie Parker',
56: 'Charlie Parker',
57: 'Charlie Parker',
58: 'Charlie Parker',
59: 'Charlie Parker',
60: 'Charlie Parker',
61: 'Charlie Parker',
62: 'Charlie Parker',
63: 'Charlie Parker',
64: 'Charlie Parker',
65: 'Charlie Parker',
66: 'Charlie Parker',
67: 'Charlie Parker',
68: 'Charlie Parker',
```

(continues on next page)

(continued from previous page)

```
 69: 'Charlie Shavers',
 70: 'Chet Baker',
 71: 'Chet Baker',
 72: 'Chet Baker',
 73: 'Chet Baker',
 74: 'Chet Baker',
 75: 'Chet Baker',
 76: 'Chet Baker',
 77: 'Chet Baker',
 78: 'Chris Potter',
 79: 'Chris Potter',
 80: 'Chris Potter',
 81: 'Chris Potter',
 82: 'Chris Potter',
 83: 'Chris Potter',
 84: 'Chris Potter',
 85: 'Chu Berry',
 86: 'Chu Berry',
 87: 'Clifford Brown',
 88: 'Clifford Brown',
 89: 'Clifford Brown',
 90: 'Clifford Brown',
 91: 'Clifford Brown',
 92: 'Clifford Brown',
 93: 'Clifford Brown',
 94: 'Clifford Brown',
 95: 'Clifford Brown',
 96: 'Coleman Hawkins',
 97: 'Coleman Hawkins',
 98: 'Coleman Hawkins',
 99: 'Coleman Hawkins',
100: 'Coleman Hawkins',
101: 'Coleman Hawkins',
102: 'Curtis Fuller',
103: 'Curtis Fuller',
104: 'David Liebman',
105: 'David Liebman',
106: 'David Liebman',
107: 'David Liebman',
108: 'David Liebman',
109: 'David Liebman',
110: 'David Liebman',
111: 'David Liebman',
112: 'David Liebman',
113: 'David Liebman',
114: 'David Liebman',
115: 'David Murray',
116: 'David Murray',
117: 'David Murray',
118: 'David Murray',
119: 'David Murray',
120: 'David Murray',
121: 'Dexter Gordon',
122: 'Dexter Gordon',
123: 'Dexter Gordon',
124: 'Dexter Gordon',
125: 'Dexter Gordon',
```

(continues on next page)

(continued from previous page)

```
126: 'Dexter Gordon',
127: 'Dickie Wells',
128: 'Dickie Wells',
129: 'Dickie Wells',
130: 'Dickie Wells',
131: 'Dickie Wells',
132: 'Dickie Wells',
133: 'Dizzy Gillespie',
134: 'Dizzy Gillespie',
135: 'Dizzy Gillespie',
136: 'Dizzy Gillespie',
137: 'Dizzy Gillespie',
138: 'Dizzy Gillespie',
139: 'Don Byas',
140: 'Don Byas',
141: 'Don Byas',
142: 'Don Byas',
143: 'Don Byas',
144: 'Don Byas',
145: 'Don Byas',
146: 'Don Byas',
147: 'Don Ellis',
148: 'Don Ellis',
149: 'Don Ellis',
150: 'Don Ellis',
151: 'Don Ellis',
152: 'Don Ellis',
153: 'Eric Dolphy',
154: 'Eric Dolphy',
155: 'Eric Dolphy',
156: 'Eric Dolphy',
157: 'Eric Dolphy',
158: 'Eric Dolphy',
159: 'Fats Navarro',
160: 'Fats Navarro',
161: 'Fats Navarro',
162: 'Fats Navarro',
163: 'Fats Navarro',
164: 'Fats Navarro',
165: 'Freddie Hubbard',
166: 'Freddie Hubbard',
167: 'Freddie Hubbard',
168: 'Freddie Hubbard',
169: 'Freddie Hubbard',
170: 'Freddie Hubbard',
171: 'George Coleman',
172: 'Gerry Mulligan',
173: 'Gerry Mulligan',
174: 'Gerry Mulligan',
175: 'Gerry Mulligan',
176: 'Gerry Mulligan',
177: 'Gerry Mulligan',
178: 'Hank Mobley',
179: 'Hank Mobley',
180: 'Hank Mobley',
181: 'Hank Mobley',
182: 'Harry Edison',
```

(continues on next page)

(continued from previous page)

```
183: 'Henry Allen',
184: 'Herbie Hancock',
185: 'Herbie Hancock',
186: 'Herbie Hancock',
187: 'Herbie Hancock',
188: 'Herbie Hancock',
189: 'J.C. Higginbotham',
190: 'J.J. Johnson',
191: 'J.J. Johnson',
192: 'J.J. Johnson',
193: 'J.J. Johnson',
194: 'J.J. Johnson',
195: 'J.J. Johnson',
196: 'J.J. Johnson',
197: 'J.J. Johnson',
198: 'Joe Henderson',
199: 'Joe Henderson',
200: 'Joe Henderson',
201: 'Joe Henderson',
202: 'Joe Henderson',
203: 'Joe Henderson',
204: 'Joe Henderson',
205: 'Joe Henderson',
206: 'Joe Lovano',
207: 'Joe Lovano',
208: 'Joe Lovano',
209: 'Joe Lovano',
210: 'Joe Lovano',
211: 'Joe Lovano',
212: 'Joe Lovano',
213: 'Joe Lovano',
214: 'John Abercrombie',
215: 'John Coltrane',
216: 'John Coltrane',
217: 'John Coltrane',
218: 'John Coltrane',
219: 'John Coltrane',
220: 'John Coltrane',
221: 'John Coltrane',
222: 'John Coltrane',
223: 'John Coltrane',
224: 'John Coltrane',
225: 'John Coltrane',
226: 'John Coltrane',
227: 'John Coltrane',
228: 'John Coltrane',
229: 'John Coltrane',
230: 'John Coltrane',
231: 'John Coltrane',
232: 'John Coltrane',
233: 'John Coltrane',
234: 'John Coltrane',
235: 'Johnny Dodds',
236: 'Johnny Dodds',
237: 'Johnny Dodds',
238: 'Johnny Dodds',
239: 'Johnny Dodds',
```

(continues on next page)

(continued from previous page)

```
240: 'Johnny Dodds',
241: 'Johnny Hodges',
242: 'Johnny Hodges',
243: 'Joshua Redman',
244: 'Joshua Redman',
245: 'Joshua Redman',
246: 'Joshua Redman',
247: 'Joshua Redman',
248: 'Kai Winding',
249: 'Kenny Dorham',
250: 'Kenny Dorham',
251: 'Kenny Dorham',
252: 'Kenny Dorham',
253: 'Kenny Dorham',
254: 'Kenny Dorham',
255: 'Kenny Dorham',
256: 'Kenny Garrett',
257: 'Kenny Garrett',
258: 'Kenny Wheeler',
259: 'Kenny Wheeler',
260: 'Kenny Wheeler',
261: 'Kid Ory',
262: 'Kid Ory',
263: 'Kid Ory',
264: 'Kid Ory',
265: 'Kid Ory',
266: 'Lee Konitz',
267: 'Lee Konitz',
268: 'Lee Konitz',
269: 'Lee Konitz',
270: 'Lee Konitz',
271: 'Lee Konitz',
272: 'Lee Konitz',
273: 'Lee Konitz',
274: 'Lee Morgan',
275: 'Lee Morgan',
276: 'Lee Morgan',
277: 'Lee Morgan',
278: 'Lester Young',
279: 'Lester Young',
280: 'Lester Young',
281: 'Lester Young',
282: 'Lester Young',
283: 'Lester Young',
284: 'Lester Young',
285: 'Lionel Hampton',
286: 'Lionel Hampton',
287: 'Lionel Hampton',
288: 'Lionel Hampton',
289: 'Lionel Hampton',
290: 'Lionel Hampton',
291: 'Louis Armstrong',
292: 'Louis Armstrong',
293: 'Louis Armstrong',
294: 'Louis Armstrong',
295: 'Louis Armstrong',
296: 'Louis Armstrong',
```

(continues on next page)

(continued from previous page)

```
297: 'Louis Armstrong',
298: 'Louis Armstrong',
299: 'Michael Brecker',
300: 'Michael Brecker',
301: 'Michael Brecker',
302: 'Michael Brecker',
303: 'Michael Brecker',
304: 'Michael Brecker',
305: 'Michael Brecker',
306: 'Michael Brecker',
307: 'Michael Brecker',
308: 'Michael Brecker',
309: 'Miles Davis',
310: 'Miles Davis',
311: 'Miles Davis',
312: 'Miles Davis',
313: 'Miles Davis',
314: 'Miles Davis',
315: 'Miles Davis',
316: 'Miles Davis',
317: 'Miles Davis',
318: 'Miles Davis',
319: 'Miles Davis',
320: 'Miles Davis',
321: 'Miles Davis',
322: 'Miles Davis',
323: 'Miles Davis',
324: 'Miles Davis',
325: 'Miles Davis',
326: 'Miles Davis',
327: 'Miles Davis',
328: 'Milt Jackson',
329: 'Milt Jackson',
330: 'Milt Jackson',
331: 'Milt Jackson',
332: 'Milt Jackson',
333: 'Milt Jackson',
334: 'Nat Adderley',
335: 'Nat Adderley',
336: 'Ornette Coleman',
337: 'Ornette Coleman',
338: 'Ornette Coleman',
339: 'Ornette Coleman',
340: 'Ornette Coleman',
341: 'Pat Martino',
342: 'Pat Metheny',
343: 'Pat Metheny',
344: 'Pat Metheny',
345: 'Pat Metheny',
346: 'Paul Desmond',
347: 'Paul Desmond',
348: 'Paul Desmond',
349: 'Paul Desmond',
350: 'Paul Desmond',
351: 'Paul Desmond',
352: 'Paul Desmond',
353: 'Paul Desmond',
```

(continues on next page)

(continued from previous page)

```
354: 'Pepper Adams',
355: 'Pepper Adams',
356: 'Pepper Adams',
357: 'Pepper Adams',
358: 'Pepper Adams',
359: 'Phil Woods',
360: 'Phil Woods',
361: 'Phil Woods',
362: 'Phil Woods',
363: 'Phil Woods',
364: 'Phil Woods',
365: 'Red Garland',
366: 'Rex Stewart',
367: 'Roy Eldridge',
368: 'Roy Eldridge',
369: 'Roy Eldridge',
370: 'Roy Eldridge',
371: 'Roy Eldridge',
372: 'Roy Eldridge',
373: 'Sidney Bechet',
374: 'Sidney Bechet',
375: 'Sidney Bechet',
376: 'Sidney Bechet',
377: 'Sidney Bechet',
378: 'Sonny Rollins',
379: 'Sonny Rollins',
380: 'Sonny Rollins',
381: 'Sonny Rollins',
382: 'Sonny Rollins',
383: 'Sonny Rollins',
384: 'Sonny Rollins',
385: 'Sonny Rollins',
386: 'Sonny Rollins',
387: 'Sonny Rollins',
388: 'Sonny Rollins',
389: 'Sonny Rollins',
390: 'Sonny Rollins',
391: 'Sonny Stitt',
392: 'Sonny Stitt',
393: 'Sonny Stitt',
394: 'Sonny Stitt',
395: 'Sonny Stitt',
396: 'Sonny Stitt',
397: 'Stan Getz',
398: 'Stan Getz',
399: 'Stan Getz',
400: 'Stan Getz',
401: 'Stan Getz',
402: 'Stan Getz',
403: 'Steve Coleman',
404: 'Steve Coleman',
405: 'Steve Coleman',
406: 'Steve Coleman',
407: 'Steve Coleman',
408: 'Steve Coleman',
409: 'Steve Coleman',
410: 'Steve Coleman',
```

(continues on next page)

(continued from previous page)

```
 411: 'Steve Coleman',
 412: 'Steve Coleman',
 413: 'Steve Lacy',
 414: 'Steve Lacy',
 415: 'Steve Lacy',
 416: 'Steve Lacy',
 417: 'Steve Lacy',
 418: 'Steve Lacy',
 419: 'Steve Turre',
 420: 'Steve Turre',
 421: 'Steve Turre',
 422: 'Von Freeman',
 423: 'Warne Marsh',
 424: 'Warne Marsh',
 425: 'Warne Marsh',
 426: 'Wayne Shorter',
 427: 'Wayne Shorter',
 428: 'Wayne Shorter',
 429: 'Wayne Shorter',
 430: 'Wayne Shorter',
 431: 'Wayne Shorter',
 432: 'Wayne Shorter',
 433: 'Wayne Shorter',
 434: 'Wayne Shorter',
 435: 'Wayne Shorter',
 436: 'Woody Shaw',
 437: 'Woody Shaw',
 438: 'Woody Shaw',
 439: 'Woody Shaw',
 440: 'Woody Shaw',
 441: 'Woody Shaw',
 442: 'Woody Shaw',
 443: 'Woody Shaw',
 444: 'Wynton Marsalis',
 445: 'Wynton Marsalis',
 446: 'Wynton Marsalis',
 447: 'Wynton Marsalis',
 448: 'Wynton Marsalis',
 449: 'Wynton Marsalis',
 450: 'Wynton Marsalis',
 451: 'Zoot Sims',
 452: 'Zoot Sims',
 453: 'Zoot Sims',
 454: 'Zoot Sims',
 455: 'Zoot Sims',
 456: 'Zoot Sims'}
```

We can now use this dictionary to create a new column `performer` in the `solos` DataFrame.

[28]:
```
solos["performer"] = solos["melid"].map(mapper)
```

[29]:
```
solos.head()
```

[29]:

	eventid	melid	onset	pitch	duration	period	division	bar	beat	\
0	1	1	10.343492	65.0	0.138776	4	1	0	1	
1	2	1	10.637642	63.0	0.171247	4	4	0	2	
2	3	1	10.843719	58.0	0.081270	4	4	0	2	

(continues on next page)

(continued from previous page)

```
3        4      1  10.948209   61.0  0.235102       4         1    0     3
4        5      1  11.232653   63.0  0.130612       4         1    0     4

   tatum  ...  loud_max   loud_med   loud_sd loud_relpos  loud_cent  loud_s2b  \
0      1  ...  0.126209  66.526087  5.541147    0.307692   0.389466  1.056169
1      1  ...  0.349751  69.133321  2.912412    0.250000   0.468687  1.120317
2      4  ...  0.094051  66.352130  3.564563    0.428571   0.531354  1.310389
3      1  ...  0.521187  66.484173  2.414298    0.818182   0.559333  0.984047
4      1  ...  0.560737  71.699054  2.185794    0.166667   0.438973  1.061262

    f0_range  f0_freq_hz  f0_med_dev   performer
0  37.794261   12.932532   -0.328442  Art Pepper
1   6.365930    6.956935   11.135423  Art Pepper
2  68.010392         NaN   32.366787  Art Pepper
3  15.443906    5.867151   -3.374696  Art Pepper
4  11.444363    8.329975    6.377737  Art Pepper

[5 rows x 27 columns]
```

[30]:
```
solos.tail()
```

[30]:
```
        eventid  melid      onset  pitch  duration  period  division  bar  \
200804   200805    456  63.135057   57.0  0.168345       4         2   53
200805   200806    456  63.303401   55.0  0.087075       4         3   54
200806   200807    456  63.390476   57.0  0.191565       4         3   54
200807   200808    456  63.640091   59.0  0.406349       4         1   54
200808   200809    456  64.058050   52.0  1.433832       4         2   54

        beat  tatum  ...  loud_max   loud_med   loud_sd loud_relpos  \
200804     4      2  ...  1.113380  72.169552  6.896394    0.687500
200805     1      1  ...  0.491496  69.732265  1.814723    0.500000
200806     1      2  ...  1.187058  76.628621  2.628726    0.411765
200807     2      1  ...  0.972676  66.042058  3.690577    0.000000
200808     3      2  ...  0.368321  58.174931  9.418678    0.053030

        loud_cent  loud_s2b    f0_range  f0_freq_hz  f0_med_dev  performer
200804   0.581956  1.271747  191.074095   10.966972  -11.891698  Zoot Sims
200805   0.595212  1.339060   40.375449         NaN  -99.173779  Zoot Sims
200806   0.590950  1.432802  104.823845   11.148561   -2.911604  Zoot Sims
200807   0.334937  1.082549  165.810976    2.659723   14.311001  Zoot Sims
200808   0.400571  1.278890   66.932198    2.153916   -9.381310  Zoot Sims

[5 rows x 27 columns]
```

4.1 Melodic arc?

Does the melodic arc also appear in the Jazz solos?

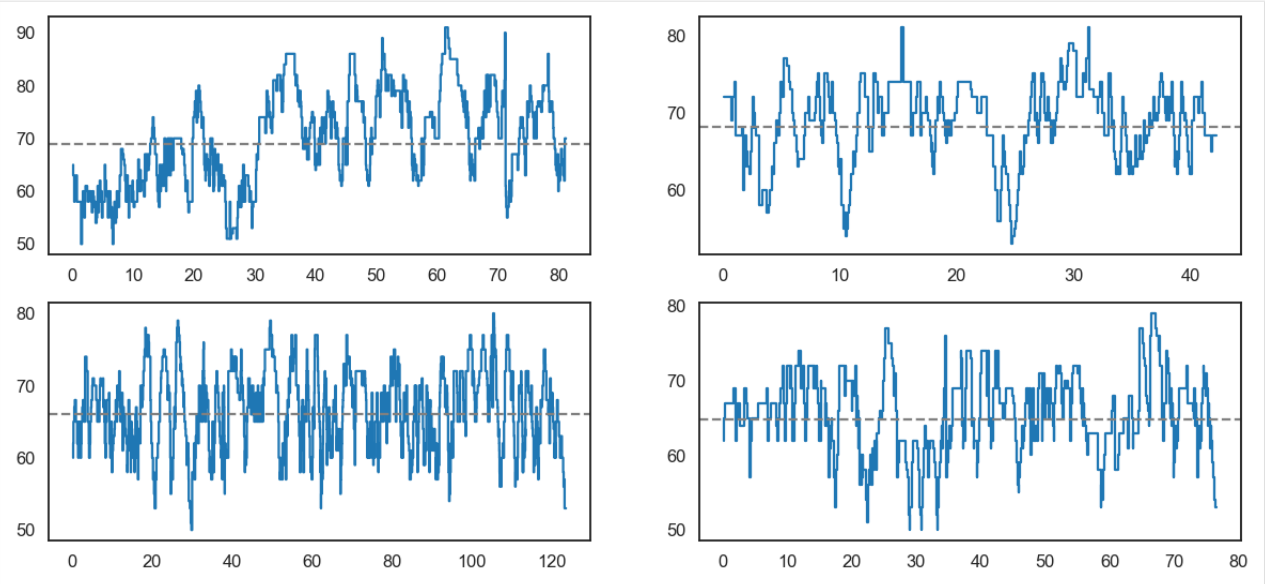
[31]:
```
def notelist(melid):

    solo = solos[solos["melid"] == melid]

    solo = solo[["pitch", "duration"]]
    solo["onset"] = solo["duration"].cumsum()
    return solo
```

[32]:
```
notelist(1)
```

[32]:
```
     pitch  duration      onset
0     65.0  0.138776   0.138776
1     63.0  0.171247   0.310023
2     58.0  0.081270   0.391293
3     61.0  0.235102   0.626395
4     63.0  0.130612   0.757007
..     ...       ...        ...
525   66.0  0.137143  80.645238
526   65.0  0.101587  80.746825
527   63.0  0.104490  80.851315
528   62.0  0.110295  80.961610
529   70.0  0.187211  81.148821

[530 rows x 3 columns]
```

[33]:
```
def plot_melodic_profile(notelist, ax=None, c=None, mean=False, Z=False,␣
↪sections=False, standardized=False):

    if ax == None:
        ax = plt.gca()

    if standardized:
        x = notelist["Rel. Onset"]
        y = notelist["Rel. MIDI Pitch"]
    else:
        x = notelist["onset"]
        y = notelist["pitch"]

    ax.step(x,y, color=c)

    if mean:
        ax.axhline(y.mean(), color="gray", linestyle="--")

    if sections:
        for l in [ x.max() * i for i in [ 1/4, 1/2, 3/4] ]:
            ax.axvline(l, color="gray", linewidth=1, linestyle="--")
```

[34]:
```
fig, axes = plt.subplots(2,2, figsize=(20,9))
axes = axes.flatten()

plot_melodic_profile(notelist(1), ax=axes[0], mean=True)
plot_melodic_profile(notelist(77), ax=axes[1], mean=True)
plot_melodic_profile(notelist(50), ax=axes[2], mean=True)
plot_melodic_profile(notelist(233), ax=axes[3], mean=True)
```



[35]:
```
def standardize(notelist):
    """
    Takes a notelist as input and returns a standardized version.
    """

    notelist["Rel. MIDI Pitch"] = (notelist["pitch"] - notelist["pitch"].mean()) /␣
↪notelist["pitch"].std()
    notelist["Rel. Duration"] = notelist["duration"] / notelist["duration"].sum()
    notelist["Rel. Onset"] = notelist["onset"] / notelist["onset"].max()

    return notelist
```

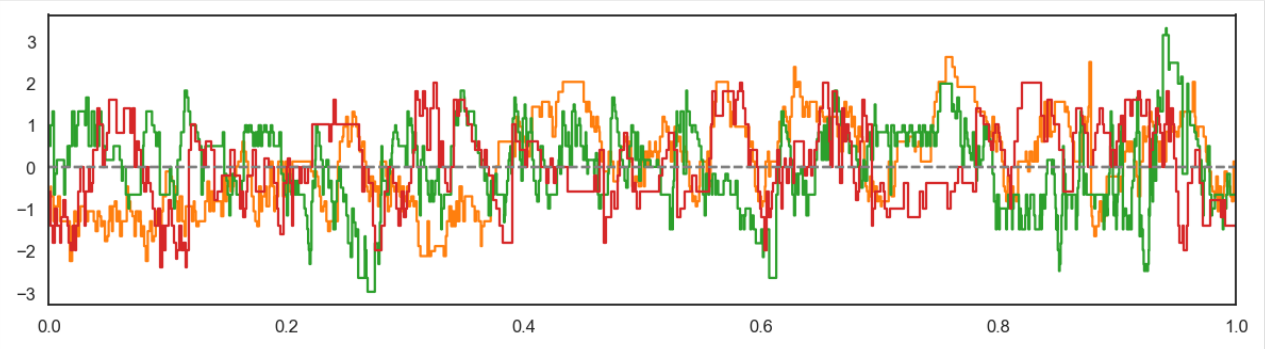
[37]:
```
standardize(notelist(1))
```

[37]:

	pitch	duration	onset	Rel. MIDI Pitch	Rel. Duration	Rel. Onset
0	65.0	0.138776	0.138776	-0.460594	0.001710	0.001710
1	63.0	0.171247	0.310023	-0.697714	0.002110	0.003820
2	58.0	0.081270	0.391293	-1.290513	0.001001	0.004822
3	61.0	0.235102	0.626395	-0.934833	0.002897	0.007719
4	63.0	0.130612	0.757007	-0.697714	0.001610	0.009329
..	...	...	...	...	...	...
525	66.0	0.137143	80.645238	-0.342034	0.001690	0.993794
526	65.0	0.101587	80.746825	-0.460594	0.001252	0.995046
527	63.0	0.104490	80.851315	-0.697714	0.001288	0.996334
528	62.0	0.110295	80.961610	-0.816274	0.001359	0.997693
529	70.0	0.187211	81.148821	0.132205	0.002307	1.000000

[530 rows x 6 columns]

[51]:
```
fig, ax = plt.subplots(figsize=(20,5))

for i in range(4):
    plot_melodic_profile(standardize(notelist(i)),
                         mean=True,
                         standardized=True)
plt.xlim(0,1)
plt.show()
```



[41]:
```
big_df = pd.concat([standardize(notelist(i)) for i in range(solos_meta.shape[0])])
```

[44]:
```
solos
```

[44]:

	eventid	melid	onset	pitch	duration	period	division	bar	\
0	1	1	10.343492	65.0	0.138776	4	1	0	
1	2	1	10.637642	63.0	0.171247	4	4	0	
2	3	1	10.843719	58.0	0.081270	4	4	0	
3	4	1	10.948209	61.0	0.235102	4	1	0	
4	5	1	11.232653	63.0	0.130612	4	1	0	
...	...	...	...	...	...	...	...	...	
200804	200805	456	63.135057	57.0	0.168345	4	2	53	
200805	200806	456	63.303401	55.0	0.087075	4	3	54	
200806	200807	456	63.390476	57.0	0.191565	4	3	54	
200807	200808	456	63.640091	59.0	0.406349	4	1	54	
200808	200809	456	64.058050	52.0	1.433832	4	2	54	

	beat	tatum	...	loud_max	loud_med	loud_sd	loud_relpos	\
0	1	1	...	0.126209	66.526087	5.541147	0.307692	
1	2	1	...	0.349751	69.133321	2.912412	0.250000	
2	2	4	...	0.094051	66.352130	3.564563	0.428571	
3	3	1	...	0.521187	66.484173	2.414298	0.818182	
4	4	1	...	0.560737	71.699054	2.185794	0.166667	
...	...	...	...	...	...	...	...	
200804	4	2	...	1.113380	72.169552	6.896394	0.687500	
200805	1	1	...	0.491496	69.732265	1.814723	0.500000	
200806	1	2	...	1.187058	76.628621	2.628726	0.411765	
200807	2	1	...	0.972676	66.042058	3.690577	0.000000	
200808	3	2	...	0.368321	58.174931	9.418678	0.053030	

	loud_cent	loud_s2b	f0_range	f0_freq_hz	f0_med_dev	performer
0	0.389466	1.056169	37.794261	12.932532	-0.328442	Art Pepper
1	0.468687	1.120317	6.365930	6.956935	11.135423	Art Pepper
2	0.531354	1.310389	68.010392	NaN	32.366787	Art Pepper
3	0.559333	0.984047	15.443906	5.867151	-3.374696	Art Pepper
4	0.438973	1.061262	11.444363	8.329975	6.377737	Art Pepper
...	...	...	...	...	...	...
200804	0.581956	1.271747	191.074095	10.966972	-11.891698	Zoot Sims
200805	0.595212	1.339060	40.375449	NaN	-99.173779	Zoot Sims
200806	0.590950	1.432802	104.823845	11.148561	-2.911604	Zoot Sims
200807	0.334937	1.082549	165.810976	2.659723	14.311001	Zoot Sims
200808	0.400571	1.278890	66.932198	2.153916	-9.381310	Zoot Sims

[200809 rows x 27 columns]

[43]:
```
big_df
```

[43]:
```
        pitch  duration      onset  Rel. MIDI Pitch  Rel. Duration  Rel. Onset
0        65.0  0.138776   0.138776        -0.460594       0.001710    0.001710
1        63.0  0.171247   0.310023        -0.697714       0.002110    0.003820
2        58.0  0.081270   0.391293        -1.290513       0.001001    0.004822
3        61.0  0.235102   0.626395        -0.934833       0.002897    0.007719
4        63.0  0.130612   0.757007        -0.697714       0.001610    0.009329
...       ...       ...        ...              ...            ...         ...
200585   62.0  0.870748  68.588934         0.014206       0.012540    0.987794
200586   57.0  0.133515  68.722449        -0.896471       0.001923    0.989717
200587   62.0  0.139320  68.861769         0.014206       0.002006    0.991723
200588   61.0  0.133515  68.995283        -0.167930       0.001923    0.993646
200589   60.0  0.441179  69.436463        -0.350065       0.006354    1.000000

[200590 rows x 6 columns]
```

[62]:
```
solos_meta["performer"].unique()
```

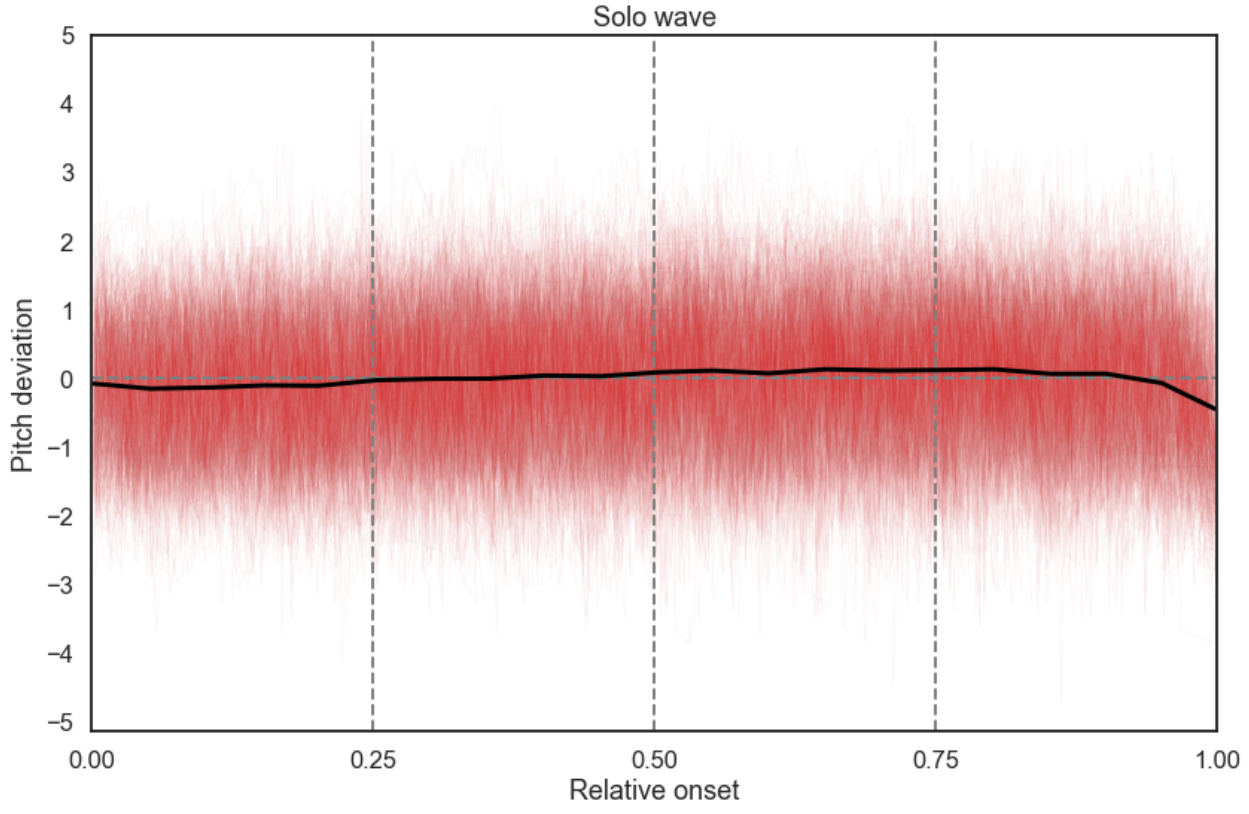
[62]:
```
array(['Art Pepper', 'Benny Carter', 'Benny Goodman', 'Ben Webster',
       'Bix Beiderbecke', 'Bob Berg', 'Branford Marsalis', 'Buck Clayton',
       'Cannonball Adderley', 'Charlie Parker', 'Charlie Shavers',
       'Chet Baker', 'Chris Potter', 'Chu Berry', 'Clifford Brown',
       'Coleman Hawkins', 'Curtis Fuller', 'David Liebman',
       'David Murray', 'Dexter Gordon', 'Dickie Wells', 'Dizzy Gillespie',
       'Don Byas', 'Don Ellis', 'Eric Dolphy', 'Fats Navarro',
       'Freddie Hubbard', 'George Coleman', 'Gerry Mulligan',
       'Hank Mobley', 'Harry Edison', 'Henry Allen', 'Herbie Hancock',
       'J.C. Higginbotham', 'J.J. Johnson', 'Joe Henderson', 'Joe Lovano',
       'John Abercrombie', 'John Coltrane', 'Johnny Dodds',
       'Johnny Hodges', 'Joshua Redman', 'Kai Winding', 'Kenny Dorham',
       'Kenny Garrett', 'Kenny Wheeler', 'Kid Ory', 'Lee Konitz',
       'Lee Morgan', 'Lester Young', 'Lionel Hampton', 'Louis Armstrong',
       'Michael Brecker', 'Miles Davis', 'Milt Jackson', 'Nat Adderley',
       'Ornette Coleman', 'Pat Martino', 'Pat Metheny', 'Paul Desmond',
       'Pepper Adams', 'Phil Woods', 'Red Garland', 'Rex Stewart',
       'Roy Eldridge', 'Sidney Bechet', 'Sonny Rollins', 'Sonny Stitt',
       'Stan Getz', 'Steve Coleman', 'Steve Lacy', 'Steve Turre',
       'Von Freeman', 'Warne Marsh', 'Wayne Shorter', 'Woody Shaw',
       'Wynton Marsalis', 'Zoot Sims'], dtype=object)
```

[67]:
```
%%time

fig, ax = plt.subplots(figsize=(12,8))

artists = ["Louis Armstrong"]

# for i, (artist, group) in enumerate(solos.groupby("performer")):
#     if artist in artists:
#         for j, group in group.groupby("melid"):
#             solo = standardize(notelist(j))
#             x = solo["Rel. Onset"]
#             y = solo["Rel. MIDI Pitch"]
#             ax. plot(x,y, lw=.5, c="tab:red", alpha=.5)

for ID in range(solos_meta.shape[0]):
    solo = standardize(notelist(ID))
    x = solo["Rel. Onset"]
```

(continues on next page)

(continued from previous page)

```
    y = solo["Rel. MIDI Pitch"]
    ax. plot(x,y, lw=.5, c="tab:red", alpha=.05)

ax.axvline(.25, lw=2, ls="--", c="gray")
ax.axvline(.5, lw=2, ls="--", c="gray")
ax.axvline(.75, lw=2, ls="--", c="gray")
ax.axhline(0, lw=2, ls="--", c="gray")

lowess = sm.nonparametric.lowess
big_x = big_df["Rel. Onset"]
big_y = big_df["Rel. MIDI Pitch"]
big_z = lowess(big_y, big_x, frac=1/10, delta=1/20)
ax.plot(big_z[:,0], big_z[:,1], c="black", lw=3)

plt.title("Solo wave")
plt.xlabel("Relative onset")
plt.ylabel("Pitch deviation")
plt.xticks(np.linspace(0,1,5))
plt.yticks(np.linspace(-5,5,11))
plt.xlim(0,1)

plt.tight_layout()
plt.savefig("img/jazz_melodic_arc.png")
plt.show()
```




```
Wall time: 7.76 s
```

4.2 Pitch vs loudness

Above we have already analyzed some melodic profiles and seen that, on average, the Jazz solos tend not to follow the melodic arch on a global scale. Now, we ask whether the pitch of the notes in the solos are related to another important feature of performance: loudness. The WJazzD contains several measures for loudness (compare the columns in the `solos` DataFrame). Here, we focus on the "Median loudness" which is stored in the `loud_med` column.

Let us look at an example.

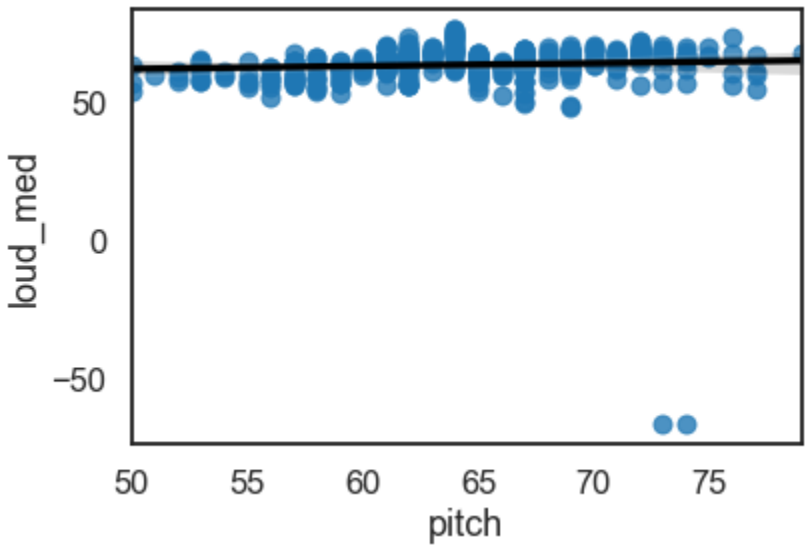
[68]:
```
example_solo = solos[ solos["melid"] == 233 ][["pitch", "loud_med"]]
```

[69]:
```
example_solo
```

[69]:
```
        pitch   loud_med
115075   62.0  67.082700
115076   65.0  65.345677
115077   67.0  66.323539
115078   69.0  69.204257
115079   62.0  69.059581
...       ...        ...
115549   59.0  61.083907
115550   57.0  58.345887
115551   55.0  65.132786
115552   54.0  59.595735
115553   53.0  58.390132

[479 rows x 2 columns]
```

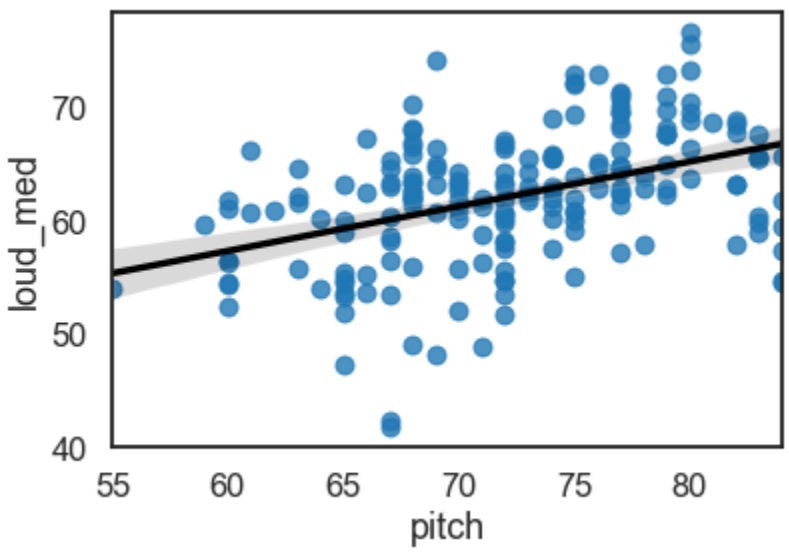
We can get a visual impression of whether there might be a direct relation between the two features by plotting it and drawing a regression line. For this, the `regplot()` function of the `seaborn` library is well-suited.

[71]:
```
sns.regplot(data=example_solo, x="pitch", y="loud_med", line_kws={"color":"black"});
```



There seems to be no clear relation; no matter how high the pitch, the loudness stays more or less the same. Let's look at another example!

[73]:
```
example_solo2 = solos[ solos["melid"] == 333 ][["pitch", "loud_med"]]

sns.regplot(data=example_solo2, x="pitch", y="loud_med", line_kws={"color":"black"});
```

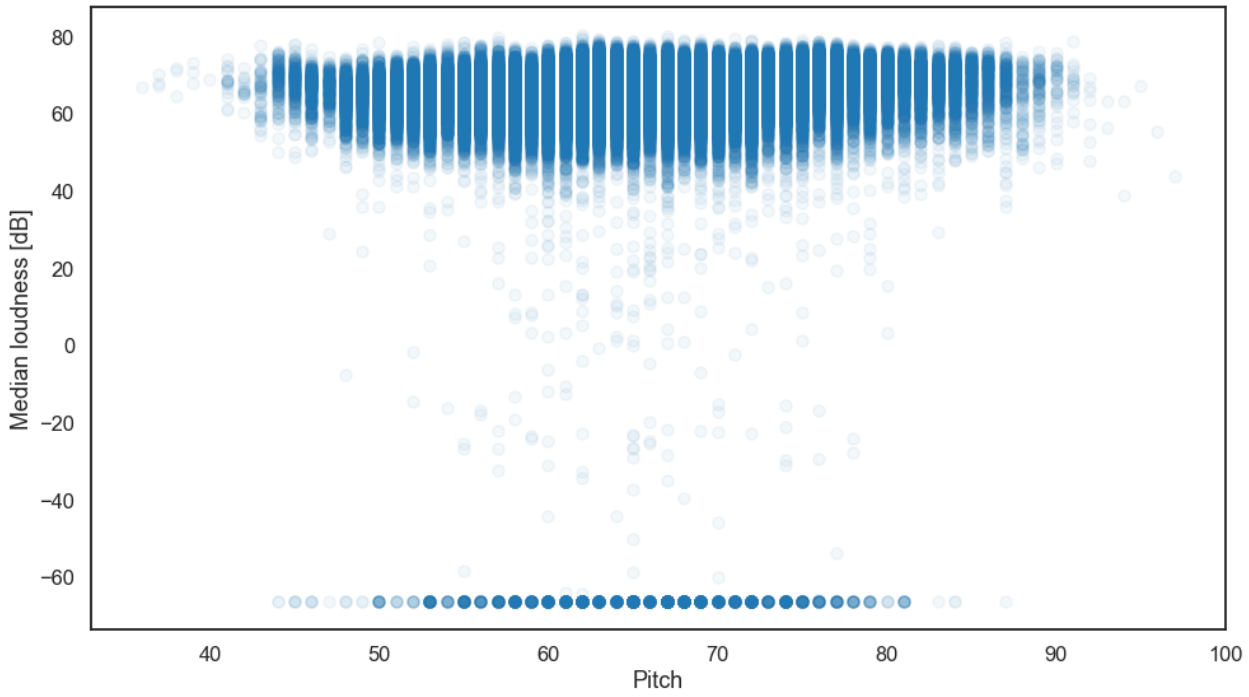


In this case, there is a positive trend. The higher the pitch, the louder the performer plays. Since we have now two different examples - in one case no relation, in the other case a positive correlation - we should now look at whether there is a trend emerging from all solos taken together.

4.3 The "rain cloud" of Jazz solos

We now take all 200'809 notes from all solos and look at the relation between their pitch and their median loudness.

[75]:
```
X = solos[["pitch", "loud_med"]].values
x = X[:,0]
y = X[:,1]

fig, ax = plt.subplots(figsize=(16,9))
ax.scatter(x,y, alpha=0.05)

plt.xlabel("Pitch")
plt.ylabel("Median loudness [dB]")
plt.show()
```



The visual impression is that of a cloud from which rain drops down and forms a puddle. Which trends can we observe?

4.4 Comparing performers

Taking all pieces together was not really informative. Maybe a somewhat closer look brings more to the front. Let us some specific performers whose solos we want to compare.

[76]:
```
selected_performers = ["Charlie Parker", "Miles Davis", "Louis Armstrong", "Herbie␣
↪Hancock", "Von Freeman", "Red Garland"]
```

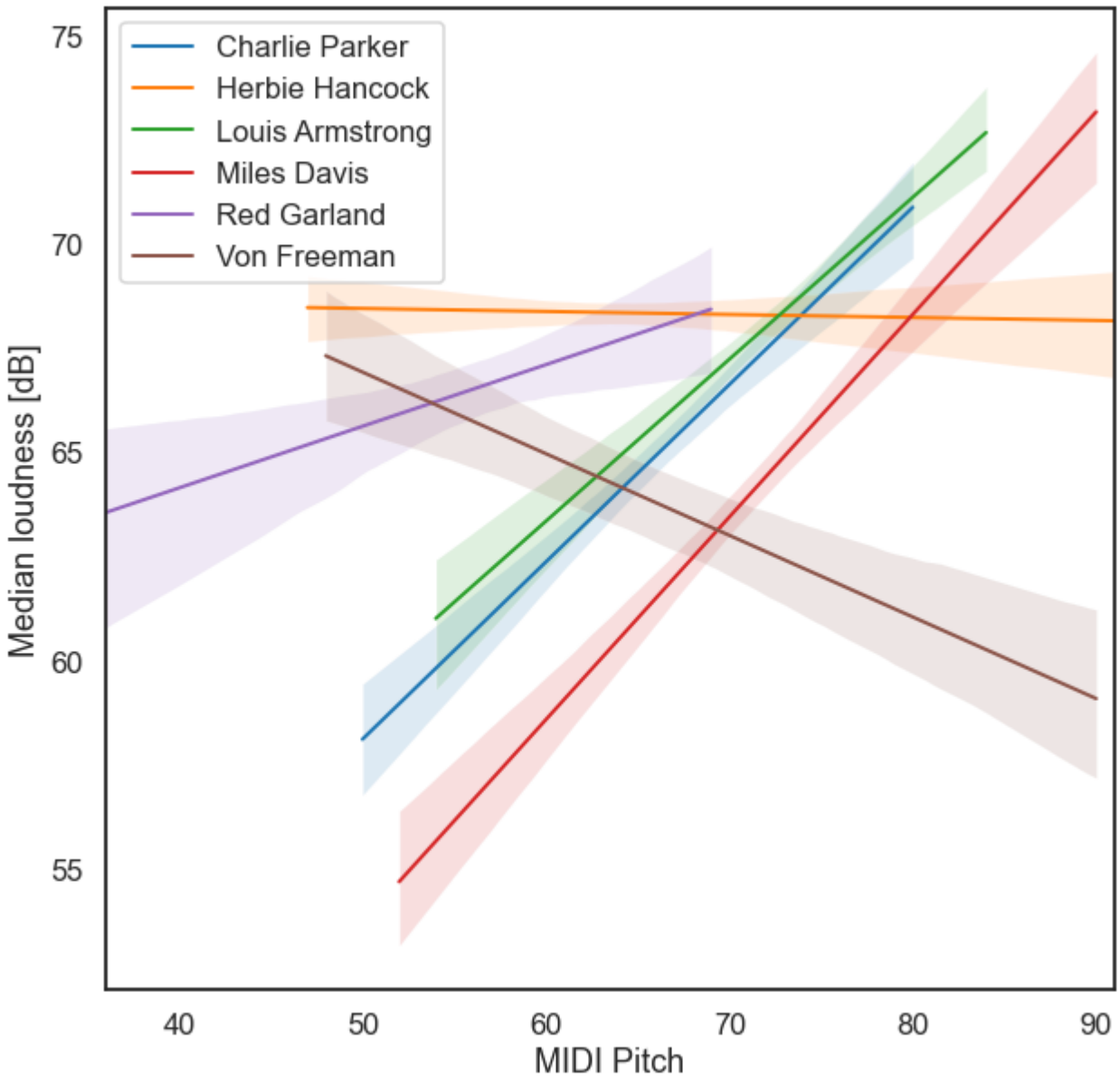
[78]:
```
grouped_df = solos.groupby("performer")
```

[79]:
```
fig, ax = plt.subplots(figsize=(10,10))

for performer, df in grouped_df:
    if performer in selected_performers:
        sns.regplot(
            data=df,
            x="pitch",
            y="loud_med",
            x_jitter=.1,
            y_jitter=.1,
            scatter_kws={"alpha":.01, "color":"grey"},
            line_kws={"lw":2},
            label=performer,
            scatter=False,
            ax=ax
        )
```

(continues on next page)

(continued from previous page)

```
plt.xlabel("MIDI Pitch")
plt.ylabel("Median loudness [dB]")
plt.legend()
plt.show()
```



Observations:

1. Most performers increase loudness with increasing pitch.
2. Charlie Parker (sax) and Louis Armstrong (t) show very similar patterns but Armstrong is generally higher.
3. Miles Davis (t) is similar to the two but plays generally softer than both.
4. Von Freeman (sax) strongly and Herbie Hancock (p) weakly decrease loudness with increasing pitch (almost all other performers show positive correlations).

5. Red Garland (p) plays generally lower than Herbie Hancock (p) but does show a positive correlation between pitch and loudness (NB: there is only one solo in the database).

Does this tell us something about performer styles or about instruments?

CHAPTER

FIVE

EXERCISE I: THE ANNOTATED BEETHOVEN CORPUS AND THE PANDAS LIBRARY

In the first exercise, you learn how to interact with `pandas` library that is very useful for deadling with tabular data. Moreover, you will have a first look at the data that we work with in the next session, the *Annotated Beethoven Corpus* (ABC). The ABC contains harmonic annotations for all string quartets by Ludwig van Beethoven. The corpus is described in

- Neuwirth, M., Harasim, D., Moss, F. C., & Rohrmeier, M. (2018). The Annotated Beethoven Corpus (ABC): A Dataset of Harmonic Analyses of All Beethoven String Quartets. *Frontiers in Digital Humanities*. https://doi.org/10.3389/fdigh.2018.00016

A study based on this corpus is

- Moss, F. C., Harasim, D., Neuwirth, M., & Rohrmeier, M. (2019). Statistical characteristics of tonal harmony: A corpus study of Beethoven's string quartets. *PLOS ONE*. https://doi.org/10.1371/journal.pone.0217242

Preparation

1. Select one member of your team who shares her/his screen (maybe someone with *some* programming experience).
2. Do all the exercises in that person's notebook together.

If you are not sure how to solve a task, Google is your friend. If you have any questions at any point, don't hesitate to ask me or Sebastian for help.

First, import the `pandas` library in the cell below. It is customary to abbreviate it with `pd`. If you are not sure how to do this, have a look at the "10 minutes to pandas" tutorial or ask Google.

[1]:
```
import pandas as pd
```

You can check which version of `pandas` you are using by using the `.__version__` attribute of `pd`(two underscores on each side).

[ ]:
```

```

The best way to learn how `pandas` works is to use it! The ABC is stored at the following address:

[2]:
```
url = "https://raw.githubusercontent.com/DCMLab/ABC/master/data/all_annotations.tsv"
```

The most important object in `pandas` is a **DataFrame** which is basically just another word for a table.

We can load the ABC from the address that is stored in the variable `url` by passing it to the `.read_table()` method. Assign it to a variable called `abc`. The code for doing this is `abc = pd.read_table(url)`.

[6]:
```
abc = pd.read_csv(url, sep='\t')
```

[7]:
```
abc.tail()
```

[7]:
```
       chord altchord  measure  beat  totbeat timesig  op  no  mov  length  \
28090      I      NaN      171   1.0    542.5     2/2  95  11    4     4.0
28091     V7      NaN      172   1.0    546.5     2/2  95  11    4     4.0
28092      I      NaN      173   1.0    550.5     2/2  95  11    4     4.0
28093      V      NaN      174   1.0    554.5     2/2  95  11    4     4.0
28094  I\\\\      NaN      175   1.0    558.5     2/2  95  11    4     4.0

      global_key local_key pedal numeral form  figbass changes relativeroot  \
28090      false         I   NaN       I  NaN      NaN     NaN          NaN
28091      false         I   NaN       V  NaN      7.0     NaN          NaN
28092      false         I   NaN       I  NaN      NaN     NaN          NaN
28093      false         I   NaN       V  NaN      NaN     NaN          NaN
28094      false         I   NaN       I  NaN      NaN     NaN          NaN

       phraseend
28090      False
28091      False
28092      False
28093      False
28094       True
```

Call the variable `abc` in the next cell.

[ ]:
```

```

As you can see, `pandas` displays the first and the last five rows, and a lot of columns of the table. A very handy way to know how large the table is, is to use the `.shape` attribute of the `abc` DataFrame. It returns a pair `(number_of_rows, number_of_columns)`. Use this attribute in the cell below and compare it to the numbers of rows and columns that are displayed directly below the DataFrame.

[ ]:
```

```

Now, try to figure out what the following columns could mean and write it to the code (like the example for `chord`).

[ ]:
```
column_meanings = """
chord: 'A chord symbol'
measure:
beat:
totbeat:
timesig:
op:
no:
mov:
length:
global_key:
local_key:
numeral:
figbass:
"""
```

We can look at individual columns of the DataFrame by **selecting** them. For example, if we had a DataFrame called `df` with a column named `temperature`, we could look at only this column by writing `df["temperature"]`. In the cell below, select the `chord` column of the DataFrame `abc`.

[ ]:
```

```

Again, we see only the first and last five entries.

We know now how many chords are annotated in this corpus and how we can select individual columns. But how many *different* chords are there? `pandas` has a very useful method for this task, called `.value_counts()`. If you select the `chord` column in the DataFrame `abc` and append the `.value_counts()` method to it, it will show you the number of times each chord appears in the corpus in descending order.

[ ]:

What are the five most common chords? Can you interpret it?

Now, in the big table that we have stored in the variable `abc` there are *all* chords in *all* of Beethoven's string quartets. What if we wanted to look only at a single of these quartets, for example the string quartet no. 11, op. 95 in F minor? The following code shows how we would do it:

```
abc[ abc["op"] == 95 ]
```

This reads as "Show me the `abc` where the `op` column of the `abc` is equal to 95 (note the double ==). In the cell below, show only the rows of the table that belong to the string quartet no. 10, op. 74 in E♭ major.

[ ]:

How could we now count all the chords in op. 74?

[ ]:

How many different chords are in op. 74? (Hint: You don't need to write code for that, the `.value_counts()` method already states it.

Above we have seen that the `global_key` contains the key of the entire piece, while `local_key` contains the keys to which a piece **modulates**. If we wanted to know which chords occur how often *either* in major *or* minor keys, we have to select them explicitly, similarly to how we selected an opus number.

However, the DataFrame does not contain a `mode` column that would contain whether a chord occurs in a major or in a minor segment. But we can work around that! Let us first see, which local keys there are in all string quartets. This can be achieved by first selecting the `local_key` column and then calling the `.unique()` method on it.

[ ]:

Apparently, there is a great variety of keys in Beethoven's string quartes. If we only want to select the major keys, we have to select all keys that start with an uppercase letter, e.g. `'VI'` **or** that start with a flat (♭, here represented by the letter `b`) **and** have uppercase letters following, for example `'bIV'`. This is a quite complicated condition. For tasks like this, one can use so-called **regular expressions** (not necessary to know exactly what this means yet). The condition mentioned above would be expressed as follows:

[ ]:
```
major_condition = "^b?[A-Z]"
```

The `^b?` part means "has the letter 'b' at the beginning but only once", and the `[A-Z]` part means "any uppercase letter". Consequently, we can select all pieces in the major mode by

[ ]:
```
abc[ abc["local_key"].str.match(major_condition) ]
```

Take the above expression and count the chords in all major keys in the cell below.

[ ]:

How would we have to change `major_condition` so that it selects only minor keys? Save the condition for minor keys in a variable `minor_condition` in the cell below.

[ ]:

Now, we are in the position to select all minor-key segments from the string quartets and count the chords!

[ ]:

Compare the first couple of chords in major and minor. Can you interpret it?

Congratulations! You completely solved the first exercise!

CHAPTER

SIX

RENAISSANCE CADENCES

This part of the course was taught by Prof. Richard Freedman, Haverford College who introduced the Lost Voices project and in particular how one can use this corpus for both close and distant reading analyses. Richard's sides can be accessed here.

CHAPTER

SEVEN

EXERCISE II: LOST VOICES CADENCE DATA

Notebook by Richard Freedman (adapted by Fabian Moss)

see https://digitalduchemin.org

As always, we start by importing a number of libraries that contain certain functions which we will use later in the analyses.

[19]:
```
import pandas as pd # working with tabular data
import altair as alt # visualization library; see https://altair-viz.github.io/
alt.renderers.enable('altair_viewer', inline=True) # to display the charts inline
import altair_viewer as av
from pathlib import Path # to handle files
import requests # to communicate with web servers
```

7.1 Get the Cadence Metadata from github as DataFrame

The cadence data from the Digital DuChemin project are available on GitHub. Conveniently, the cadence data is already stored in a CSV file that we can download and store in a table. The function below encapsulates these steps so that when we call it, it returns a dataframe with the stored cadence data.

[20]:
```
# get the data function
def get_data():
    url = "https://raw.githubusercontent.com/RichardFreedman/LostVoicesCadenceViewer/
↪main/LV_CadenceData.csv"
    cadence_data_raw = pd.read_csv(url)
    # The following adds the list of 'similar' cadences to the DF
    cadence_json =  requests.get("https://raw.githubusercontent.com/bmill42/DuChemin/
↪master/phase1/data/duchemin.similarities.json").json()
    cadence_data_raw['similarity'] = cadence_json
    return  cadence_data_raw

cadence_data_raw = get_data()
```

The variable `cadence_data_raw` now contains the cadence data. Remember that we can use the `.head()` method to desplay the first couple of rows. Display the first 10 rows in the next cell.

[21]:
```
cadence_data_raw.head(10)
```

[21]:
```
                                    phrase_number  \
0  DC0625.8: Dont nous serons plus contentz qu'eulx.
1         DC0625.6: Ami perfaict plus que les dieux,
2  DC0625.5: Mais ne fault point que doubte on face,
```

(continues on next page)

(continued from previous page)

```
3        DC0625.2: Non plus que j'ay de bonne grace,
4                DC0625.1: Si je n'avois de fermeté,
5  DC0624.10: Par ton amour si fais cesser la mie...
6  DC0624.8: C'est pour t'aimer pour Dieu il t'en...
7  DC0624.6: Qui ne penetre au plus profond de l'...
8     DC0624.2: Je me suis mis comme ami en debvoir,
9   DC0623.8: Mais la vostre est par peur estaincte.

                                composition_number cadence_role_cantz  \
0                DC0625: Si je n'avois de fermeté                   S
1                DC0625: Si je n'avois de fermeté                   S
2                DC0625: Si je n'avois de fermeté                None
3                DC0625: Si je n'avois de fermeté                   S
4                DC0625: Si je n'avois de fermeté                   T
5  DC0624: Si tu as veu que pour ton feu estaindre                  S
6  DC0624: Si tu as veu que pour ton feu estaindre                  T
7  DC0624: Si tu as veu que pour ton feu estaindre                None
8  DC0624: Si tu as veu que pour ton feu estaindre                   T
9           DC0623: Vous souvient-il ; ma mignonne                   S

                 cadence_alter cadence_final_tone cadence_role_tenz  \
0                          NaN                 G                 T
1                          NaN                 A                 T
2                          NaN                 D              None
3                          NaN                 G                 T
4  Displaced, Inverted, Evaded                 C                Ct
5                          NaN                 G                 T
6  Displaced, Inverted, Evaded                 C                Ct
7                          NaN                 G              None
8              Inverted, Evaded                C                 S
9                          NaN                 G                 T

  cadence_kind cadence_final_tone_before cadence_final_tone_after  \
0    Authentic                         A                     None
1    Authentic                         D                        G
2       Plagal                         G                        A
3    Authentic                         C                        D
4    Authentic                      None                        G
5    Authentic                         C                     None
6    Authentic                         G                        G
7       Plagal                         C                        C
8    Authentic                      None                        G
9    Authentic                         C                     None

  cadence_kind_before cadence_kind_after final_cadence  \
0           Authentic               None             G
1              Plagal          Authentic             G
2           Authentic          Authentic             G
3           Authentic             Plagal             G
4                None          Authentic             G
5           Authentic               None             G
6              Plagal          Authentic             G
7           Authentic          Authentic             G
8                None             Plagal             G
9           Authentic               None             G

                                     similarity
```

(continues on next page)

(continued from previous page)

```
0  [5, 9, 13, 15, 48, 64, 74, 77, 82, 85, 107, 11...
1                                          [87, 122]
2                                     [88, 335, 512]
3                                            [493]
4                                            [624]
5  [0, 9, 13, 15, 48, 64, 74, 77, 82, 85, 107, 11...
6                                               []
7                                            [373]
8                                               []
9  [0, 5, 13, 15, 48, 64, 74, 77, 82, 85, 107, 11...
```

Do you understand what kind of information each column contains? Interpret the contents of the following columns:

- `final_cadence`
- `cadence_final_tone`
- `cadence_role_cantz` / `cadence_role_tenz`
- `cadence_kind` / `cadence_kind_before` / `cadence_kind_after`
- `similarity` (looking back at the definition of the `get_data` function might help)

7.1.1 Get "Counts" for Values of a Particular Column

Use the `value_counts()` method to count the occurrences in the `cadence_kind` column of the DataFrame. Remember that you can select a column of a DataFrame like this: `df[col_name]` where `df` is the name of the variable that stores the DataFrame and `col_name` is the name of the column. Which cadence types are most frequent in this repertoire? Do you understand each cadence type?

[22]:
```
cadence_data_raw['cadence_kind'].value_counts()
```

[22]:
```
Authentic    882
Plagal       158
Phrygian      98
CadInCad      37
CAD NDLT      20
Name: cadence_kind, dtype: int64
```

7.1.2 Sort Data According to Selected Columns

Use the `.sort_values()` method to sort the raw cadence data by the final tone of the cadence **and** by the candence kind. Remember that this method takes either a column name or a list of column names as input.

The `.sort_values()` method is documented here: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html As described there, you can reverse the ascending order (default) of the sorted values by changing a keyword to the method. Store the result in a variable named `Sorted` (uppercase S!) and inspect its first 5 rows.

[23]:
```
Sorted = cadence_data_raw.sort_values(['cadence_final_tone','cadence_kind'],␣
↪ascending=False)
Sorted.head(5)
```

[23]:
```
                                        phrase_number  \
703            DC0511.3: Et vous tuez le corps et l'ame
1109      DC0114.9: Elle me paist ceste seconde Heleine
```

(continues on next page)

(continued from previous page)

```
7     DC0624.6: Qui ne penetre au plus profond de l'...
16     DC0621.5: Ne pensez pas que ferois tel deffault?
66              DC0609.7: Demander le fault à mon cœur,

                                 composition_number cadence_role_cantz  \
703                  DC0511: Ce disoit une jeune dame                B
1109   DC0114: Trop justement je forme une complainte                S
7     DC0624: Si tu as veu que pour ton feu estaindre             None
16          DC0621: Un gay berger prioit une bergiere             None
66               DC0609: Je ne suis devin ne prophète             None

            cadence_alter cadence_final_tone cadence_role_tenz cadence_kind  \
703   Inverted, Displaced               None                Ct    Authentic
1109    Displaced, Evaded               None                Ct    Authentic
7                     NaN                  G              None       Plagal
16                    NaN                  G              None       Plagal
66                    NaN                  G              None       Plagal

     cadence_final_tone_before cadence_final_tone_after cadence_kind_before  \
703                          D                        D           Authentic
1109                         C                        A           Authentic
7                            C                        C           Authentic
16                           G                        G           Authentic
66                     B-flat                         G           Authentic

     cadence_kind_after final_cadence           similarity
703            Phrygian             G                   []
1109          Authentic             A                   []
7             Authentic             G                [373]
16            Authentic             G  [66, 132, 335, 512]
66            Authentic             G                 [16]
```

`Sorted` now contains the phrases sorted according to the tone of the final cadence and the kind of the cadence. As you can see, there are a lot of repeating values adjacent to each other.

A better overview is obtained by **grouping** the data with the `.groupby()` method. Like `.value_counts()` this method takes a column name or a list of column names as input.

7.2 Grouping

Group the data by the same two columns as before and store the result in a variable `grouped`.

[24]:
```
grouped = cadence_data_raw.groupby(['cadence_final_tone', 'final_cadence'])
```

What happens if you want to inspect the grouped data?

[25]:
```
grouped
```

[25]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000225571E6C10>
```

Most likely, you didn't see anything. That is because you didn't yet specify in which way the grouped data should be displayed. Here, we would like to count the cadences. Thus, we apply the `.count()` method to the grouped data.

[26]:
```
grouped.count()
```

[26]:

cadence_final_tone	final_cadence	phrase_number	composition_number	cadence_role_cantz	cadence_alter
A	A	30	30	30	25
	B-flat	1	1	0	0
	C	2	2	2	0
	D	48	48	47	29
	E	6	6	6	2
	F	26	26		
	G	27	27		
B-flat	A	1	1		
	B-flat	4	4		
	D	8	8		
	F	2	2		
	G	68	68		
C	A	11	11		
	B-flat	1	1		
	C	54	54		
	D	5	5		
	E	4	4		
	F	72	72		
	G	44	44		
D	A	2	2		
	B-flat	6	6		
	C	1	1		
	D	89	89		
	E	1	1		
	F	11	11		
	G	167	167		
E	A	11	11		
	C	4	4		
	D	4	4		
	E	4	4		
	F	1	1		
	G	7	7		
F	B-flat	1	1		
	C	9	9		
	D	14	14		
	E	1	1		
	F	136	136		
	G	5	5		
G	A	7	7		
	B-flat	7	7		
	C	28	28		
	D	23	23		
	E	1	1		
	F	14	14		
	G	225	225		
None	A	1	1		
	G	1	1		

(continues on next page)

(continued from previous page)

```
                    F                                25              15
                    G                                27              16
B-flat              A                                 1               1
                    B-flat                            4               3
                    D                                 8               5
                    F                                 2               2
                    G                                68              45
C                   A                                11              10
                    B-flat                            1               1
                    C                                52              32
                    D                                 5               3
                    E                                 4               2
                    F                                71              64
                    G                                44              32
D                   A                                 2               1
                    B-flat                            5               5
                    C                                 1               0
                    D                                85              56
                    E                                 1               1
                    F                                10               7
                    G                               159              98
E                   A                                10               9
                    C                                 4               2
                    D                                 4               1
                    E                                 4               2
                    F                                 1               1
                    G                                 6               3
F                   B-flat                            1               1
                    C                                 9               3
                    D                                14              11
                    E                                 1               1
                    F                               136             104
                    G                                 5               5
G                   A                                 7               7
                    B-flat                            7               3
                    C                                28              17
                    D                                21              11
                    E                                 1               1
                    F                                14               8
                    G                               222             106
None                A                                 1               1
                    G                                 1               1

                                        cadence_role_tenz  cadence_kind  \
cadence_final_tone final_cadence
A                   A                                30            30
                    B-flat                            0             1
                    C                                 2             2
                    D                                47            48
                    E                                 6             6
                    F                                25            26
                    G                                27            27
B-flat              A                                 1             1
                    B-flat                            4             4
                    D                                 8             8
                    F                                 2             2
                    G                                68            68
```

(continues on next page)

(continued from previous page)

```
C                   A                                 11             11
                    B-flat                             1              1
                    C                                 51             54
                    D                                  5              5
                    E                                  4              4
                    F                                 71             72
                    G                                 44             44
D                   A                                  2              2
                    B-flat                             5              6
                    C                                  1              1
                    D                                 85             89
                    E                                  1              1
                    F                                 10             11
                    G                                159            167
E                   A                                 11             11
                    C                                  4              4
                    D                                  4              4
                    E                                  4              4
                    F                                  1              1
                    G                                  6              7
F                   B-flat                             1              1
                    C                                  9              9
                    D                                 14             14
                    E                                  1              1
                    F                                136            136
                    G                                  5              5
G                   A                                  7              7
                    B-flat                             7              7
                    C                                 28             28
                    D                                 21             23
                    E                                  1              1
                    F                                 14             14
                    G                                222            225
None                A                                  1              1
                    G                                  1              1

                                        cadence_final_tone_before  \
cadence_final_tone final_cadence
A                   A                                      30
                    B-flat                                  1
                    C                                       2
                    D                                      48
                    E                                       6
                    F                                      26
                    G                                      27
B-flat              A                                       1
                    B-flat                                  4
                    D                                       8
                    F                                       2
                    G                                      68
C                   A                                      11
                    B-flat                                  1
                    C                                      54
                    D                                       5
                    E                                       4
                    F                                      72
                    G                                      44
```

(continues on next page)

(continued from previous page)

```
D                   A                                    2
                    B-flat                               6
                    C                                    1
                    D                                   89
                    E                                    1
                    F                                   11
                    G                                  167
E                   A                                   11
                    C                                    4
                    D                                    4
                    E                                    4
                    F                                    1
                    G                                    7
F                   B-flat                               1
                    C                                    9
                    D                                   14
                    E                                    1
                    F                                  136
                    G                                    5
G                   A                                    7
                    B-flat                               7
                    C                                   28
                    D                                   23
                    E                                    1
                    F                                   14
                    G                                  225
None                A                                    1
                    G                                    1

                                       cadence_final_tone_after  \
cadence_final_tone final_cadence
A                   A                                   30
                    B-flat                               1
                    C                                    2
                    D                                   48
                    E                                    6
                    F                                   26
                    G                                   27
B-flat              A                                    1
                    B-flat                               4
                    D                                    8
                    F                                    2
                    G                                   68
C                   A                                   11
                    B-flat                               1
                    C                                   54
                    D                                    5
                    E                                    4
                    F                                   72
                    G                                   44
D                   A                                    2
                    B-flat                               6
                    C                                    1
                    D                                   89
                    E                                    1
                    F                                   11
                    G                                  167
```

(continues on next page)

(continued from previous page)

```
E                  A                                 11
                   C                                  4
                   D                                  4
                   E                                  4
                   F                                  1
                   G                                  7
F                  B-flat                             1
                   C                                  9
                   D                                 14
                   E                                  1
                   F                                136
                   G                                  5
G                  A                                  7
                   B-flat                             7
                   C                                 28
                   D                                 23
                   E                                  1
                   F                                 14
                   G                                225
None               A                                  1
                   G                                  1

                                 cadence_kind_before  cadence_kind_after  \
cadence_final_tone final_cadence
A                  A                              30                  30
                   B-flat                          1                   1
                   C                               2                   2
                   D                              48                  48
                   E                               6                   6
                   F                              26                  26
                   G                              27                  27
B-flat             A                               1                   1
                   B-flat                          4                   4
                   D                               8                   8
                   F                               2                   2
                   G                              68                  68
C                  A                              11                  11
                   B-flat                          1                   1
                   C                              54                  54
                   D                               5                   5
                   E                               4                   4
                   F                              72                  72
                   G                              44                  44
D                  A                               2                   2
                   B-flat                          6                   6
                   C                               1                   1
                   D                              89                  89
                   E                               1                   1
                   F                              11                  11
                   G                             167                 167
E                  A                              11                  11
                   C                               4                   4
                   D                               4                   4
                   E                               4                   4
                   F                               1                   1
                   G                               7                   7
F                  B-flat                          1                   1
```

(continues on next page)

(continued from previous page)

```
                    C                                9                9
                    D                               14               14
                    E                                1                1
                    F                              136              136
                    G                                5                5
G                   A                                7                7
                    B-flat                           7                7
                    C                               28               28
                    D                               23               23
                    E                                1                1
                    F                               14               14
                    G                              225              225
None                A                                1                1
                    G                                1                1

                                        similarity
cadence_final_tone final_cadence
A                   A                           30
                    B-flat                       1
                    C                            2
                    D                           48
                    E                            6
                    F                           26
                    G                           27
B-flat              A                            1
                    B-flat                       4
                    D                            8
                    F                            2
                    G                           68
C                   A                           11
                    B-flat                       1
                    C                           54
                    D                            5
                    E                            4
                    F                           72
                    G                           44
D                   A                            2
                    B-flat                       6
                    C                            1
                    D                           89
                    E                            1
                    F                           11
                    G                          167
E                   A                           11
                    C                            4
                    D                            4
                    E                            4
                    F                            1
                    G                            7
F                   B-flat                       1
                    C                            9
                    D                           14
                    E                            1
                    F                          136
                    G                            5
G                   A                            7
                    B-flat                       7
```

(continues on next page)

(continued from previous page)

```
                  C                  28
                  D                  23
                  E                   1
                  F                  14
                  G                 225
None              A                   1
                  G                   1
```

- Compare the result of the `.count()` method to the one you already know for displaying the first 5 rows.

[27]:
```
grouped.head()
```

[27]:
```
                                        phrase_number  \
0     DC0625.8: Dont nous serons plus contentz qu'eulx.
1            DC0625.6: Ami perfaict plus que les dieux,
2     DC0625.5: Mais ne fault point que doubte on face,
3           DC0625.2: Non plus que j'ay de bonne grace,
4                  DC0625.1: Si je n'avois de fermeté,
...                                                ...
996         DC0207.3: L’on en veit oncques de si belle,
1082  DC0118.9: Vous les ferez tous vifz crever de r...
1083   DC0118.8: Besongnez donc et de jour et de nuict.
1097     DC0116.8: As tu osé luy faire un tel oultrage?
1109      DC0114.9: Elle me paist ceste seconde Heleine

                                composition_number cadence_role_cantz  \
0                    DC0625: Si je n'avois de fermeté                 S
1                    DC0625: Si je n'avois de fermeté                 S
2                    DC0625: Si je n'avois de fermeté              None
3                    DC0625: Si je n'avois de fermeté                 S
4                    DC0625: Si je n'avois de fermeté                 T
...                                               ...               ...
996              DC0207: Joye et santé, ma demoiselle                 S
1082        DC0118: Vrais amateurs du plaisir de Vénus                 S
1083        DC0118: Vrais amateurs du plaisir de Vénus                Ct
1097  DC0116: Maistre Ambrelin, confesseur de nonettes                 B
1109    DC0114: Trop justement je forme une complainte                 S

                   cadence_alter cadence_final_tone cadence_role_tenz  \
0                            NaN                 G                 T
1                            NaN                 A                 T
2                            NaN                 D              None
3                            NaN                 G                 T
4     Displaced, Inverted, Evaded                 C                Ct
...                           ...               ...               ...
996                      Evaded                  D                 T
1082                       None                  G                 T
1083                  Displaced                  E                 B
1097          Displaced, Inverted                E                Ct
1109            Displaced, Evaded             None                Ct

     cadence_kind cadence_final_tone_before cadence_final_tone_after  \
0       Authentic                         A                     None
1       Authentic                         D                        G
2          Plagal                         G                        A
3       Authentic                         C                        D
4       Authentic                      None                        G
```

(continues on next page)

(continued from previous page)

```
...          ...                        ...                       ...
996      Phrygian                         G                         G
1082    Authentic                         E                         A
1083     Phrygian                         A                         G
1097     Phrygian                         E                         C
1109    Authentic                         C                         A

     cadence_kind_before cadence_kind_after final_cadence  \
0              Authentic               None             G
1                 Plagal          Authentic             G
2              Authentic          Authentic             G
3              Authentic             Plagal             G
4                   None          Authentic             G
...                  ...                ...           ...
996            Authentic          Authentic        B-flat
1082            Phrygian          Authentic             A
1083           Authentic          Authentic             A
1097              Plagal          Authentic             A
1109           Authentic          Authentic             A

                                              similarity
0     [5, 9, 13, 15, 48, 64, 74, 77, 82, 85, 107, 11...
1                                            [87, 122]
2                                       [88, 335, 512]
3                                                [493]
4                                                [624]
...                                                ...
996                                                []
1082                                           [1115]
1083                                               []
1097                                               []
1109                                               []

[177 rows x 13 columns]
```

- What are the main differences between these two? Describe them in the cell below.
-
-
-

7.2.1 Visualizations

In the following, you see a number of visualizations of the cadence data. For each of these figures, write down what it shows and interpret what you see (e.g. the most interesting observation).

7.3 Synoptic View of Cadence Types, Tones, and Last Cadence of Piece

[39]:
```
#Now the generic views and full data sets

all_tone_1 = alt.Chart(cadence_data_raw).mark_circle().encode(
    x='cadence_final_tone',
    y='final_cadence',
    color='cadence_kind'
).properties(
    width=400,
    title='All Cadence Tones, Types, and Finals'
)

all_tone_1.save('img/chart.html', embed_options={'renderer':'svg'})
all_tone_1
```

[39]:
```
alt.Chart(...)
```

Observations

-
-

7.4 Histograms by Cadence Type

[29]:
```
# Summary stacked bar chart listed by cadence tone

chartA = alt.Chart(cadence_data_raw).mark_bar().encode(
    alt.X("cadence_final_tone"),
    y='count()',
    color='cadence_kind'
).properties(
    width=600,
    height=300,
    title='Summary of Cadence Tones by Type'
)

chartA
```

[29]:
```
alt.Chart(...)
```

Observations

-
-

7.5 Histogram of Types by Tone

[30]:
```
# Summary stacked bar chart listed by cadence kind

chartB = alt.Chart(cadence_data_raw).mark_bar().encode(
    alt.X("cadence_kind"),
    y='count()',
    color='cadence_final_tone'
).properties(
    width=600,
    height=300,
    title='Summary of Cadence Type by Tone'
)
chartB
```

[30]:
```
alt.Chart(...)
```

Observations

-
-

7.5.1 Filter Cadences by Final Tones

[31]:
```
# here just pulling a selection of the complete dataframe--some columns

cadence_data_short = cadence_data_raw[["cadence_final_tone", "cadence_kind", "final_
↪cadence", "composition_number", "phrase_number"]]

# Create a list of possible values and multiselect menu with them in it.

cadence_list_1 = cadence_data_short['cadence_final_tone'].unique()

# Select Cadence Tones Here
# Possible values:  A, B-flat, C, D, E, F, G

cadences_selected_1 = ["B-flat"]

# Mask to filter dataframe:  returns only those "selected" in previous step
mask_cadences_1 = cadence_data_short['cadence_final_tone'].isin(cadences_selected_1)

# And now a new dataframe with only the elements that statisfy the conditions

cadence_data_masked_1 = cadence_data_short[mask_cadences_1]

# Display Result
cadence_data_masked_1.head()
```

[31]:
```
    cadence_final_tone cadence_kind final_cadence  \
51              B-flat    Authentic             G
61              B-flat       Plagal             G
```

(continues on next page)

(continued from previous page)

```
67              B-flat    Authentic             G
75              B-flat    Authentic             G
121             B-flat    Authentic             G

                                  composition_number  \
51   DC0612: Puisque voulez que de vous je m'absente
61   DC0610: Un bon vieillard qui n'avoit que le bec
67              DC0609: Je ne suis devin ne prophète
75                    DC0607: Si m'amie a de fermeté
121          DC0716: En ce verd moys, temps opportun

                                           phrase_number
51   DC0612.5: Trop plus seray satisfaict en mes pl...
61   DC0610.7: Plus de cent fois, et ne peult desla...
67        DC0609.6: Que ce qu'elle faict pour le seur,
75    DC0607.5: Mais quoy fault il que doubte en face,
121                 DC0716.5: La à gogo la serviroye,
```

Observations

-
-

7.5.2 Chart for Filtered Cadence Tones

[32]:
```
#This is for filtered tones based on settings of the previous
diagram_1 = alt.Chart(cadence_data_masked_1).mark_circle().encode(
    x='cadence_kind',
    y='composition_number',
    color='final_cadence',
    tooltip=['phrase_number', 'cadence_kind', 'final_cadence']
).properties(
    width=400,
    title='Cadence Final Tones'
)

diagram_1
```

[32]:
```
alt.Chart(...)
```

Observations

-
-

7.5.3 Filter Cadences by Kind

[33]:
```
# here just pulling a selection of the complete dataframe--some columns

cadence_data_short = cadence_data_raw[["cadence_final_tone", "cadence_kind", "final_
↪cadence", "composition_number", "phrase_number"]]
cadence_list_2 = cadence_data_short['cadence_kind'].unique()

# Select Cadence Kinds Here
# Possible values:  Authentic, Plagal, Phrygian, CadInCad, CAD NDLT

cadences_selected_2 = ["Plagal"]

# Mask to filter dataframe
mask_cadences_2 = cadence_data_short['cadence_kind'].isin(cadences_selected_2)

cadence_data_masked_2= cadence_data_short[mask_cadences_2]


cadence_data_masked_2.head()
```

[33]:
```
   cadence_final_tone cadence_kind final_cadence  \
2                   D       Plagal             G
7                   G       Plagal             G
12                  D       Plagal             G
16                  G       Plagal             G
21                  C       Plagal             F

                                composition_number  \
2                  DC0625: Si je n'avois de fermeté
7   DC0624: Si tu as veu que pour ton feu estaindre
12           DC0623: Vous souvient-il ; ma mignonne
16        DC0621: Un gay berger prioit une bergiere
21                  DC0620: En amour y a du plaisir

                                      phrase_number
2   DC0625.5: Mais ne fault point que doubte on face,
7   DC0624.6: Qui ne penetre au plus profond de l'...
12      DC0623.1: Vous souvient il point ma mignonne,
16   DC0621.5: Ne pensez pas que ferois tel deffault?
21        DC0620.7: Quicter le plaisir pour la peine,
```

7.5.4 Chart for Filtered Cadence Kinds

[34]:
```
# This diagram for filtered tones (just oned)
diagram_2 = alt.Chart(cadence_data_masked_2).mark_circle().encode(
    x='cadence_final_tone',
    y='composition_number',
    color='final_cadence',
    tooltip=['phrase_number', 'cadence_final_tone', 'final_cadence']
).properties(
    width=400,
    title='Cadence Kinds'
)

diagram_2
```

[34]:
```
alt.Chart(...)
```

7.5.5 Filter by Last Cadence of Piece

[35]:
```
# Dialogue to Select Last Cadence of Piece
#

# Create a list of possible values and multiselect menu with them in it.
# here just pulling a selection of the complete dataframe--some columns

cadence_data_short = cadence_data_raw[["cadence_final_tone", "cadence_kind", "final_
↪cadence", "composition_number", "phrase_number"]]
cadence_list_3 = cadence_data_short['final_cadence'].unique()

# Select Final Cadence Tones Kinds Here
# Select Cadence Tones Here
# Possible values:  A, B-flat, C, D, E, F, G

cadences_selected_3 = ["B-flat"]

# Mask to filter dataframe
mask_cadences_3 = cadence_data_short['final_cadence'].isin(cadences_selected_3)

cadence_data_masked_3 = cadence_data_short[mask_cadences_3]

cadence_data_masked_3.head()
```

[35]:
```
    cadence_final_tone cadence_kind final_cadence  \
378             B-flat    Authentic        B-flat
379                  G    Authentic        B-flat
380                  D       Plagal        B-flat
381                  G    Authentic        B-flat
382                  D     Phrygian        B-flat

                                 composition_number  \
378  DC0405: La grant doulceur de vostre cler visaige
379  DC0405: La grant doulceur de vostre cler visaige
380  DC0405: La grant doulceur de vostre cler visaige
381  DC0405: La grant doulceur de vostre cler visaige
```

(continues on next page)

(continued from previous page)

```
382  DC0405: La grant doulceur de vostre cler visaige

                                          phrase_number
378  DC0405.1: La grant doulceur de vostre cler vis...
379     DC0405.2: Me donne espoir de mercy recepvoir ,
380  DC0405.5: N'ayez donc plus merveilles de me ve...
381  DC0405.8: Je vis sans vie, et sans mourir je m...
382   DC0405.7: Car soubz espoir de vie et mort avoir,
```

7.5.6 Diagram by Last Cadence of Piece

[36]:
```
# This is for filtered tones (just oned)
diagram_3 = alt.Chart(cadence_data_masked_3).mark_circle().encode(
    x='cadence_final_tone',
    y='composition_number',
    color='cadence_kind',
    tooltip=['phrase_number', 'cadence_final_tone', 'final_cadence']
).properties(
    width=400,
    title='Final Cadence of Piece'
)

diagram_3
```

[36]:
```
alt.Chart(...)
```

7.5.7 Distribution of Cadence Tones, Types and Finals for Corpus

[37]:
```
# table of pieces with details via hover

tone_diagram = alt.Chart(cadence_data_raw).mark_circle().encode(
    x='cadence_kind',
    y='composition_number',
    color='final_cadence',
    tooltip=['phrase_number', 'cadence_final_tone', 'cadence_kind']
).properties(
    width=400,
    title='All Cadence Tones, Types, and Finals'
)

tone_diagram
```

[37]:
```
alt.Chart(...)
```

CHAPTER

EIGHT

DATA-DRIVEN MUSIC HISTORY

[1]:
```
import pandas as pd # for working with tabular data
pd.set_option('display.max_columns', 500)
import matplotlib.pyplot as plt # for plotting
plt.style.use("fivethirtyeight") # select specific plotting style
import seaborn as sns; sns.set_context("talk")
import numpy as np
```

8.1 Research Questions

- General: How can we study historical changes quantitatively?
- Specific: What can we say about the history of tonality based on a dataset of musical pieces?

8.2 A bit of theory

[2]:
```
note_names = list("FCGDAEB") # diatonic note names in fifths ordering
note_names
```

[2]:
```
['F', 'C', 'G', 'D', 'A', 'E', 'B']
```

[3]:
```
accidentals = ["bb", "b", "", "#", "##"] # up to two accidentals is suffient here
accidentals
```

[3]:
```
['bb', 'b', '', '#', '##']
```

[4]:
```
lof = [ n + a for a in accidentals for n in note_names ] # lof = "Line of Fifths"
print(lof)
```

```
['Fbb', 'Cbb', 'Gbb', 'Dbb', 'Abb', 'Ebb', 'Bbb', 'Fb', 'Cb', 'Gb', 'Db', 'Ab', 'Eb',
↪'Bb', 'F', 'C', 'G', 'D', 'A', 'E', 'B', 'F#', 'C#', 'G#', 'D#', 'A#', 'E#', 'B#',
↪'F##', 'C##', 'G##', 'D##', 'A##', 'E##', 'B##']
```

[5]:
```
len(lof) # how long is this line-of-fifths segment?
```

[5]:
```
35
```

We call the elements on the line of fifths **tonal pitch-classes**

8.3 Data

8.3.1 A (kind of) large corpus: TP3C

Here, we use a dataset that was specifically compiled for this kind of analysis, the Tonal pitch-class counts corpus (TP3C) (Moss, Neuwirth, Rohrmeier, 2020)

- 2,012 pieces
- 75 composers
- approx. spans 600 years of music history
- does not contain complete pieces but only counts of tonal pitch-classes

[6]:
```
import pandas as pd # to work with tabular data

url = "https://raw.githubusercontent.com/DCMLab/TP3C/master/tp3c.tsv"
data = pd.read_table(url)

data.sample(10)
```

[6]:

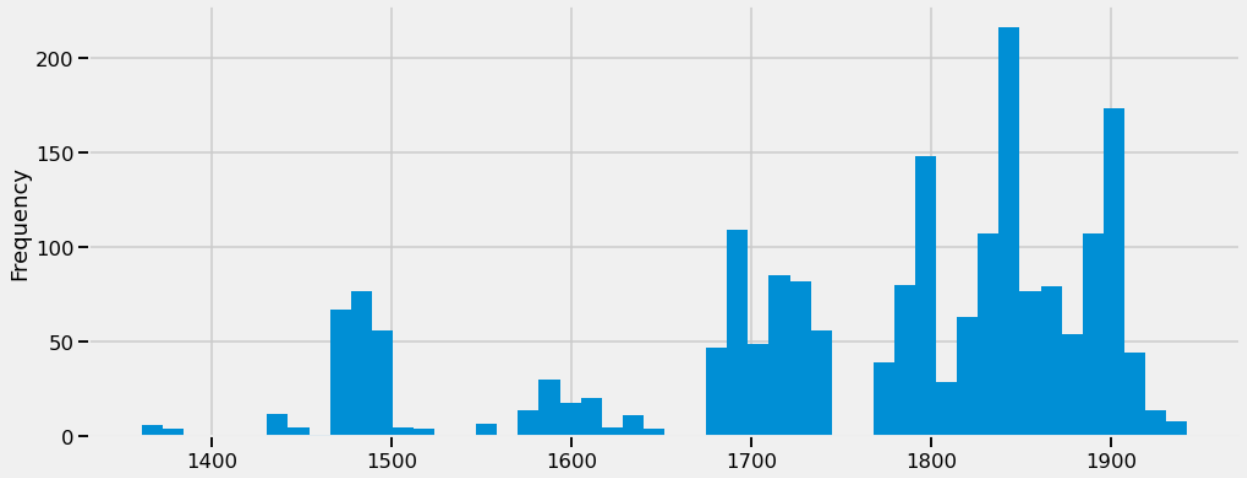
	composer	composer_first	work_group	work_catalogue	opus	no	mov	title	composition	publication	source	display_year	Fbb	Cbb	Gbb	Dbb	Abb	Ebb	Bbb	Fb	Cb	Gb	Db
1742	Corelli	Arcangelo	12 Trio Sonatas	Op.	3	4	1.0	NaN	NaN	1689.0	CCARH	1689.0	0	0	0	0	0	0	0	0	0	0	0
468	Bach	Johann Sebastian	Inventions and Sinfonias	BWV	779	NaN	NaN	NaN	NaN	1723.0	MS	1723.0	0	0	0	0	0	0	0	0	0	0	0
1630	Chopin	Frédéric	Mazurkas	Op.	33	2	NaN	NaN	1837.0	1838.0	CCARH	1837.0	0	0	0	0	0	0	0	0	0	24	16
1604	Joplin	Scott	Ragtimes	NaN	NaN	NaN	NaN	Lily Queen	NaN	1907.0	CCARH	1907.0	0	0	0	0	0	0	0	0	0	0	5
137	Bach	Johann Sebastian	Wohltemperiertes Klavier II	BWV	888	2	NaN	NaN	1740.0	NaN	MS	1740.0	0	0	0	0	0	0	0	0	0	0	0
1231	Schubert	Franz	Die schöne Müllerin	D. 795	NaN	18	NaN	Trockne Blumen	NaN	1823.0	OSLC	1823.0	0	0	0	0	0	0	0	0	0	0	0
530	Brahms	Johannes	8 Klavierstücke	Op.	76	4	NaN	Intermezzo	1878.0	NaN	DCML	1878.0	0	0	0	0	0	0	0	43	18	64	21
1311	Schumann	Robert	Dichterliebe	Op.	48	3	NaN	Die Rose, die Lilie, die Taube	1840.0	NaN	OSLC	1840.0	0	0	0	0	0	0	0	0	0	0	0
338	Bach	Johann Sebastian	Wohltemperiertes Klavier I	BWV	863	1	NaN	NaN	1722.0	NaN	MS	1722.0	0	0	0	0	0	0	0	0	0	0	0
713	Fauré	Gabriel	NaN	Op.	6	2	NaN	Tristesse	NaN	1880.0	MS	1880.0	0	0	0	0	0	0	0	0	7	0	43

(continues on next page)

(continued from previous page)

```
       Ab   Eb   Bb    F    C    G    D    A    E    B   F#   C#   G#  D#  A#  \
1742    0    0    0    0    0   33   75   47   70  107  108   56    7   8  24
468     0   17   68   98   92   77   82   81   56   17    3    7    0   0   0
1630   15   17   44   48   32  117  397  388  251  123  153  155   54   0   0
1604   23   19   51  180  353  252  134  188  162   75   55   14   17  17   7
137     0    0    0    0    3   13   69  108  101  112  105  114  101  45  16
1231    0    0    4    0   14   86   34   65  214  309  117   57  115  60  20
530    31  213  113   65   62   42   70   34    9   20    4   26    6   0   0
1311    0    0    0    0    5   52   69   61   40   45   52   48    3   1   0
338     0    0    0    0    0    0    1    9   62   72   65   90   89  81  74
713   172  126   90  174  202  169   85    0    9   68    0    1    3   1   0

      E#  B#  F##  C##  G##  D##  A##  E##  B##
1742   3   0    0    0    0    0    0    0    0
468    0   0    0    0    0    0    0    0    0
1630  10   0    0    0    0    0    0    0    0
1604   0   0    0    0    0    0    0    0    0
137   10   7    0    0    0    0    0    0    0
1231   0  16    0    0    0    0    0    0    0
530    0   0    0    0    0    0    0    0    0
1311   0   0    0    0    0    0    0    0    0
338   29  12   19    8    0    0    0    0    0
713    1   0    0    0    0    0    0    0    0
```

[8]:
```
data["display_year"].plot(kind="hist", bins=50, figsize=(15,6)); # historical overview
```



- it can be seen that there are large gaps and that some historical periods are underrepresented
- however, it is not so obvious how to fix that
- do we want a uniform distribution over time?
- do we want a “historically accurate” distribution?
- do we want to remove geographical/gender/class/instrument/etc. biases?
- on one hand, balanced datasets are likely not to reflect historical realities
- on the other hand, such datasets rather represent the “canon”, that is a contemporary selection of “valuable” compositions that may differ greatly from what was considered relevant at the time

–> There is no unique objective answer to these questions. It is important to be aware of these limitations and take

them into account when interpreting the results

For this workshop we ignore all the metadata about the pieces (titles, composer names etc.) but only focus on their tonal material. Therefore, we don't need all the columns of the table.

[9]:
```
tpc_counts = data.loc[:, lof] # select all rows (":") and the lof columns
tpc_counts.sample(20)
```

[9]:

	Fbb	Cbb	Gbb	Dbb	Abb	Ebb	Bbb	Fb	Cb	Gb	Db	Ab	Eb	Bb	\
155	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1073	0	0	0	0	0	0	0	0	11	23	83	189	219	114	
1975	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1178	0	0	0	0	0	0	0	0	0	0	0	0	11	84	
402	0	0	0	0	0	0	0	0	1	2	1	2	2	9	
98	0	0	0	0	2	0	4	2	6	41	118	144	114	164	
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
344	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1784	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
407	0	0	0	0	0	0	0	0	0	0	0	3	0	11	
1992	0	0	0	0	0	0	0	0	3	4	27	138	315	155	
543	0	0	0	0	0	0	0	0	0	17	3	0	28	61	
1046	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1032	0	0	0	0	0	1	5	4	18	35	47	132	95	205	
737	0	0	0	0	19	22	16	41	103	101	33	48	251	183	
186	0	0	0	0	0	0	0	0	0	0	4	2	16	97	
1653	0	0	0	0	0	0	0	0	0	0	1	2	0	16	
1115	0	0	0	0	0	0	0	0	0	10	8	6	41	64	
1886	0	0	0	0	0	0	0	0	0	0	6	149	233	174	
343	0	0	0	0	0	0	0	0	0	0	0	1	19	53	

	F	C	G	D	A	E	B	F#	C#	G#	D#	A#	E#	B#	F##	\
155	0	0	0	8	32	149	224	179	194	212	191	161	52	22	28	
1073	229	225	323	163	41	58	126	72	26	0	0	0	0	0	0	
1975	0	0	0	24	42	70	31	16	36	21	1	0	0	0	0	
1178	101	77	111	149	110	29	9	26	2	0	0	0	0	0	0	
402	13	100	214	103	179	347	337	250	153	83	150	87	36	18	11	
98	241	340	142	44	78	60	29	8	1	2	0	0	0	0	0	
11	52	193	249	123	54	124	103	10	0	28	4	1	0	0	0	
344	3	2	6	38	133	76	16	21	85	27	7	0	0	0	0	
1784	0	0	0	36	71	68	45	39	48	33	10	0	2	0	0	
407	148	180	168	162	111	161	105	30	5	13	0	0	0	0	0	
1992	253	327	490	223	54	54	135	58	4	0	4	0	1	0	0	
543	36	52	120	136	59	14	2	0	0	0	0	0	0	0	0	
1046	0	7	11	15	4	20	32	17	14	4	3	14	3	0	0	
1032	173	155	347	361	401	499	530	626	491	263	262	232	86	35	25	
737	97	31	9	48	92	4	1	0	0	0	0	0	0	0	0	
186	176	198	138	97	128	88	41	21	13	2	0	0	0	0	0	
1653	123	237	223	172	98	112	87	29	2	9	2	0	0	0	0	
1115	42	20	48	25	19	2	6	0	0	0	0	0	0	0	0	
1886	230	269	408	203	38	24	68	22	0	0	0	0	0	0	0	
343	102	146	102	292	549	729	483	235	377	285	109	52	9	6	0	

	C##	G##	D##	A##	E##	B##
155	13	1	0	0	0	0
1073	0	0	0	0	0	0
1975	0	0	0	0	0	0
1178	0	0	0	0	0	0
402	6	4	0	0	0	0
98	0	0	0	0	0	0

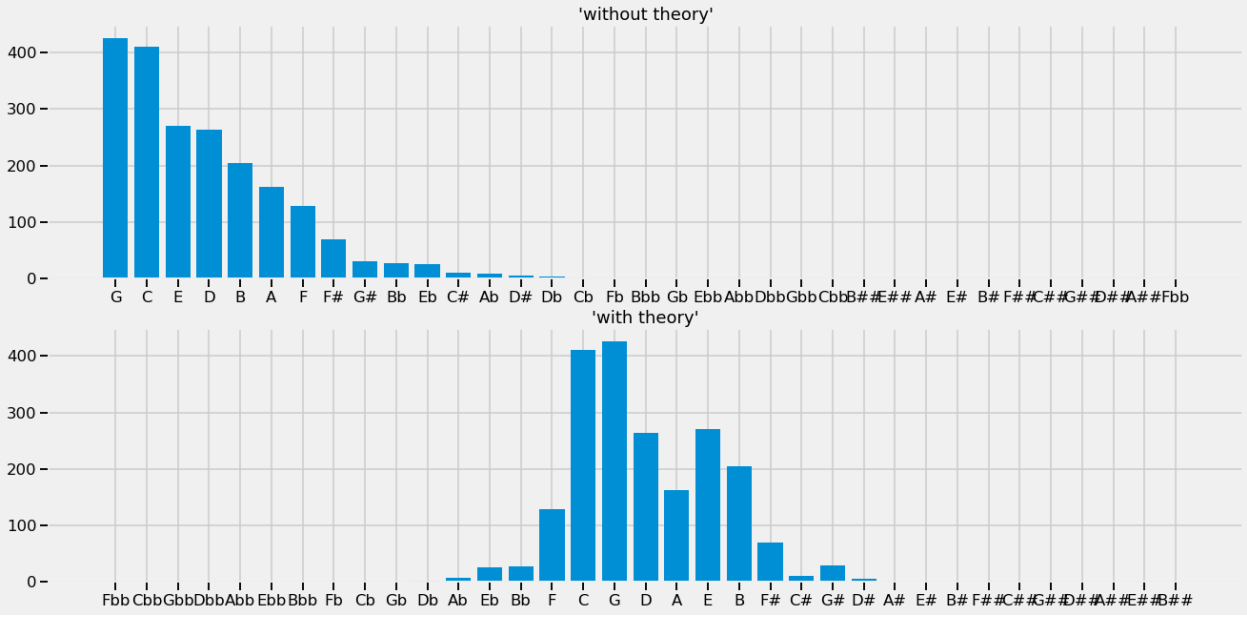
(continues on next page)

(continued from previous page)

```
11       0    0    0    0    0    0
344      0    0    0    0    0    0
1784     0    0    0    0    0    0
407      0    0    0    0    0    0
1992     0    0    0    0    0    0
543      0    0    0    0    0    0
1046     0    0    0    0    0    0
1032    26   10    2    0    0    0
737      0    0    0    0    0    0
186      0    0    0    0    0    0
1653     0    0    0    0    0    0
1115     0    0    0    0    0    0
1886     0    0    0    0    0    0
343      0    0    0    0    0    0
```

[11]:
```
piece = tpc_counts.iloc[10]

fig, axes = plt.subplots(2, 1, figsize=(20,10))

axes[0].bar(piece.sort_values(ascending=False).index, piece.sort_
↪values(ascending=False))
axes[0].set_title("'without theory'")

axes[1].bar(piece.index, piece)
axes[1].set_title("'with theory'")

# plt.savefig("img/random_piece.png")
# plt.show()
```

[11]:
```
Text(0.5, 1.0, "'with theory'")
```



Let us have an overview of the note counts in these pieces!

If we would just look at the raw counts of the tonal pitch-classe, we could not learn much from it. Using a theoretical model (the line of fifths) shows that the notes in pieces are usually come from few adjacent keys (you don't say!).

We probably have very long pieces (sonatas) and very short pieces (songs) in the dataset. Since we don't want length (or the absolute number of notes in a piece) to have an effect, we rather consider tonal pitch-class distributions instead counts, by normalizing all pieces to sum to one.

[12]:
```
tpc_dists = tpc_counts.div(tpc_counts.sum(axis=1), axis=0)
tpc_dists.sample(20)
```

[12]:

	Fbb	Cbb	Gbb	Dbb	Abb	Ebb	Bbb	Fb	Cb	\
606	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1844	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
251	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
737	0.0	0.0	0.0	0.0	0.017288	0.020018	0.014559	0.037307	0.093722	
10	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
822	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
202	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
850	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1609	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
555	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
940	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.005703	0.002852	
1165	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1438	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1548	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1265	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.005924	
1169	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1368	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1741	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
1302	0.0	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000	0.000000	
80	0.0	0.0	0.0	0.0	0.001965	0.002947	0.000000	0.008841	0.030452	

	Gb	Db	Ab	Eb	Bb	F	C	\
606	0.000000	0.000000	0.000000	0.000000	0.059524	0.059524	0.061905	
1844	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
251	0.000000	0.000000	0.000000	0.000000	0.000424	0.027107	0.044473	
737	0.091902	0.030027	0.043676	0.228389	0.166515	0.088262	0.028207	
10	0.000000	0.001466	0.003908	0.012702	0.013679	0.063019	0.200293	
822	0.000000	0.000000	0.000000	0.000000	0.022222	0.133333	0.155556	
202	0.003268	0.011438	0.024510	0.003268	0.039216	0.091503	0.207516	
850	0.000000	0.000000	0.000000	0.000000	0.000000	0.081545	0.184549	
1609	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
555	0.000000	0.000000	0.000000	0.000000	0.002088	0.083507	0.171190	
940	0.004753	0.046578	0.057034	0.125475	0.039924	0.109316	0.135932	
1165	0.000000	0.000000	0.000000	0.000000	0.000000	0.002660	0.000000	
1438	0.000000	0.000000	0.000000	0.000000	0.038251	0.077869	0.099727	
1548	0.000000	0.011111	0.011111	0.044444	0.077778	0.088889	0.088889	
1265	0.000000	0.004739	0.077014	0.158768	0.055687	0.084123	0.261848	
1169	0.011472	0.024857	0.042065	0.055449	0.082218	0.084130	0.149140	
1368	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
1741	0.000000	0.000000	0.003036	0.090081	0.178138	0.176113	0.153846	
1302	0.002725	0.013624	0.005450	0.087193	0.141689	0.040872	0.111717	
80	0.051081	0.090373	0.123772	0.140472	0.125737	0.080550	0.055992	

	G	D	A	E	B	F#	C#	\
606	0.235714	0.161905	0.147619	0.104762	0.057143	0.054762	0.045238	
1844	0.005330	0.089552	0.171642	0.173774	0.143923	0.119403	0.148188	
251	0.035578	0.077086	0.159255	0.213892	0.154172	0.058026	0.085133	
737	0.008189	0.043676	0.083712	0.003640	0.000910	0.000000	0.000000	
10	0.208109	0.128481	0.079140	0.132389	0.100147	0.034196	0.004885	
822	0.044444	0.266667	0.155556	0.133333	0.088889	0.000000	0.000000	

(continues on next page)

(continued from previous page)

```
202   0.160131  0.065359  0.096405  0.169935  0.063725  0.014706  0.013072
850   0.184549  0.180258  0.124464  0.098712  0.133047  0.008584  0.004292
1609  0.000000  0.000000  0.027883  0.093790  0.016477  0.062104  0.130545
555   0.118998  0.068894  0.204593  0.173278  0.125261  0.006263  0.004175
940   0.221483  0.135932  0.038023  0.013308  0.051331  0.010456  0.001901
1165  0.106383  0.292553  0.156915  0.079787  0.130319  0.117021  0.063830
1438  0.112022  0.140710  0.143443  0.158470  0.092896  0.068306  0.038251
1548  0.100000  0.088889  0.088889  0.088889  0.077778  0.077778  0.066667
1265  0.164692  0.082938  0.011848  0.020142  0.058057  0.014218  0.000000
1169  0.128107  0.051625  0.080306  0.057361  0.051625  0.068834  0.045889
1368  0.008929  0.041667  0.107143  0.230655  0.196429  0.117560  0.105655
1741  0.135628  0.122470  0.103239  0.012146  0.018219  0.007085  0.000000
1302  0.212534  0.171662  0.130790  0.000000  0.024523  0.046322  0.010899
80    0.049116  0.053045  0.028487  0.051081  0.057957  0.010806  0.006876

            G#        D#        A#        E#        B#        F##       C##  \
606   0.007143  0.004762  0.000000  0.000000  0.000000  0.000000  0.000000
1844  0.101279  0.023454  0.007463  0.011727  0.004264  0.000000  0.000000
251   0.087251  0.043626  0.008895  0.002541  0.002118  0.000424  0.000000
737   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
10    0.014656  0.002931  0.000000  0.000000  0.000000  0.000000  0.000000
822   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
202   0.016340  0.011438  0.008170  0.000000  0.000000  0.000000  0.000000
850   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1609  0.309252  0.177440  0.032953  0.020279  0.102662  0.016477  0.010139
555   0.033403  0.006263  0.002088  0.000000  0.000000  0.000000  0.000000
940   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1165  0.042553  0.007979  0.000000  0.000000  0.000000  0.000000  0.000000
1438  0.013661  0.002732  0.013661  0.000000  0.000000  0.000000  0.000000
1548  0.066667  0.022222  0.000000  0.000000  0.000000  0.000000  0.000000
1265  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1169  0.026769  0.013384  0.015296  0.009560  0.001912  0.000000  0.000000
1368  0.117560  0.044643  0.020833  0.005952  0.000000  0.002976  0.000000
1741  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
1302  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
80    0.016699  0.011788  0.000000  0.001965  0.000000  0.000000  0.000000

      G##  D##  A##  E##  B##
606   0.0  0.0  0.0  0.0  0.0
1844  0.0  0.0  0.0  0.0  0.0
251   0.0  0.0  0.0  0.0  0.0
737   0.0  0.0  0.0  0.0  0.0
10    0.0  0.0  0.0  0.0  0.0
822   0.0  0.0  0.0  0.0  0.0
202   0.0  0.0  0.0  0.0  0.0
850   0.0  0.0  0.0  0.0  0.0
1609  0.0  0.0  0.0  0.0  0.0
555   0.0  0.0  0.0  0.0  0.0
940   0.0  0.0  0.0  0.0  0.0
1165  0.0  0.0  0.0  0.0  0.0
1438  0.0  0.0  0.0  0.0  0.0
1548  0.0  0.0  0.0  0.0  0.0
1265  0.0  0.0  0.0  0.0  0.0
1169  0.0  0.0  0.0  0.0  0.0
1368  0.0  0.0  0.0  0.0  0.0
1741  0.0  0.0  0.0  0.0  0.0
1302  0.0  0.0  0.0  0.0  0.0
```

(continues on next page)

(continued from previous page)

```
80    0.0  0.0  0.0  0.0  0.0
```

For further numerical analysis, we extract the data from this table and assign it to a variable `X`.

[14]:
```
# extract values of table to matrix
X = tpc_dists.values

X.shape # shows (#rows, #columns) of X
```

[14]:
```
(2012, 35)
```

Now, `X` is a 2012 × 35 matrix where the rows represent the pieces and the columns (also called "features" or "dimensions") represent the relative frequency of tonal pitch-classes.

Thinking in 35 dimensions is quite difficult for most people. Without trying to imagine what this would look like, what can we already say about this data?

Since each piece is a point in this 35-D space and pieces are represented as vectors, pieces that have similar tonal pitch-class distributions must be close in this space (whatever this looks like).

What groups of pieces that cluster together? Maybe pieces of the same composer are similar to each other? Maybe pieces from a similar time? Maybe pieces for the same instruments?

If we find clusters, these would still be in 35-D and thus difficult to interpret. Luckily, there are a range of so-called *dimensionality reduction* methods that transform the data into lower-dimensional spaces so that we actually can look at them.

A very common dimensionality reduction method is **Principal Components Analysis (PCA)**.

The basic idea of PCA is:

- find dimensions in the data that maximize the variance in this direction
- these dimensions have to be orthogonal to each other (mutually independent)
- these dimensions are called the *principal components*
- each principal component is associated with how much of the data variance it explains

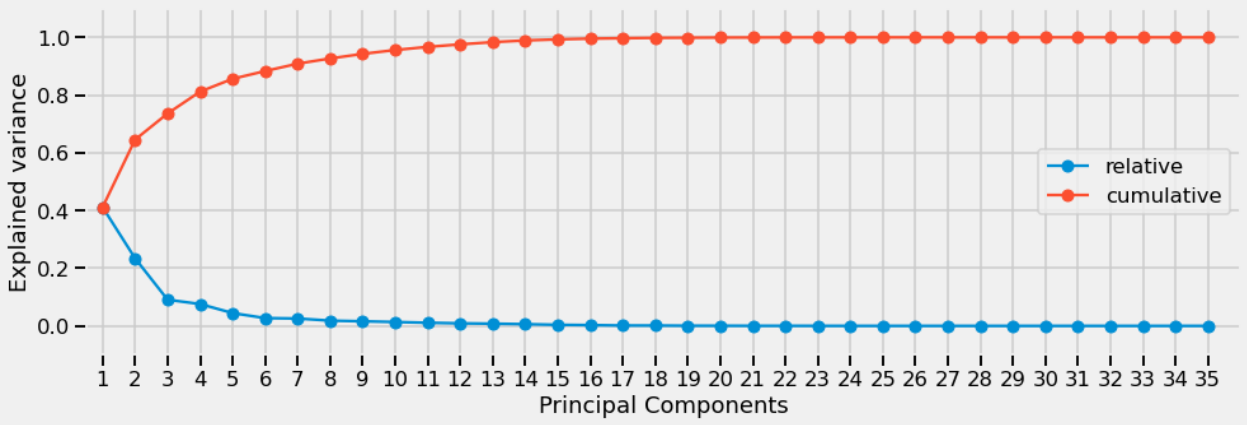
[15]:
```
import numpy as np # for numerical computations
import sklearn
from sklearn.decomposition import PCA # for dimensionality reduction

pca = sklearn.decomposition.PCA(n_components=35) # initialize PCA with 35 dimensions
pca.fit(X) # apply it to the data
variance = pca.explained_variance_ratio_ # assign explained variance to variable
```

[16]:
```
fig, ax = plt.subplots(figsize=(14,5))
x = np.arange(35)
ax.plot(x, variance, label="relative", marker="o")
ax.plot(x, variance.cumsum(), label="cumulative", marker="o")
ax.set_xlim(-0.5, 35)
ax.set_ylim(-0.1, 1.1)
ax.set_xlabel("Principal Components")
ax.set_ylabel("Explained variance")
plt.xticks(np.arange(len(lof)), np.arange(len(lof)) + 1) # because Pyhon starts␣
↪counting at 0

plt.legend(loc="center right")
plt.tight_layout()
```

(continues on next page)

(continued from previous page)

```
# plt.savefig("img/explained_variance.png")
# plt.show()
```



[17]:
```
variance[:5]
```

[17]:
```
array([0.41144591, 0.23410347, 0.09063507, 0.07574242, 0.04436989])
```

The first principal component explains 41.1% of the variance of the data, the second explains 23.4% and the third 9%. Together, this amounts to 73.6%.

Almost three quarters of the variance in the dataset is retained by reducing the dimensionality from 35 to 3 dimensions (8.6%)! If we reduce the data to two dimensions, we still can explain ≈ 65% of the variance.

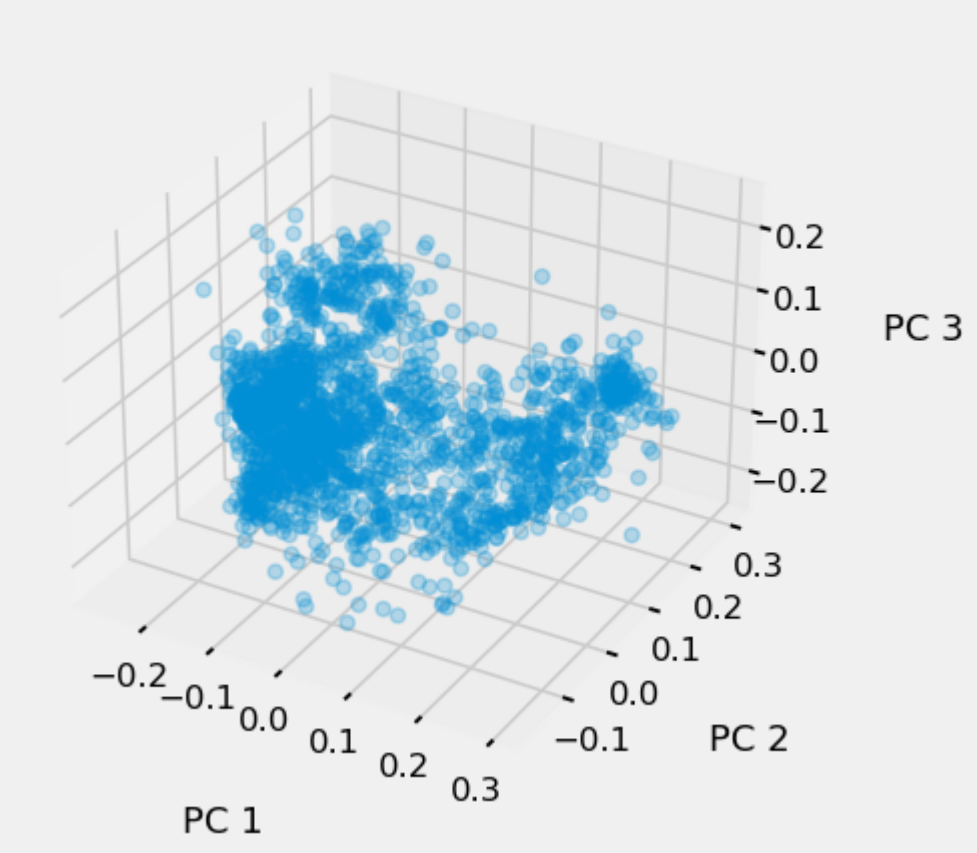
This is great because it means that we can look at the data in 2 or 3 dimensions without loosing too much information.

8.4 Recovering the line of fifths from data

[18]:
```
pca3d = PCA(n_components=3)
pca3d.fit(X)

X_ = pca3d.transform(X)
X_.shape
```

[18]:
```
(2012, 3)
```

[20]:
```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(6,6))

ax = fig.add_subplot(111, projection='3d')
ax.scatter(X_[:,0], X_[:,1], X_[:,2], s=50, alpha=.25) # c=cs,
ax.set_xlabel("PC 1", labelpad=30)
ax.set_ylabel("PC 2", labelpad=30)
ax.set_zlabel("PC 3", labelpad=30)

plt.tight_layout()
# plt.savefig("img/3d_scatter.png")
# plt.show()
```

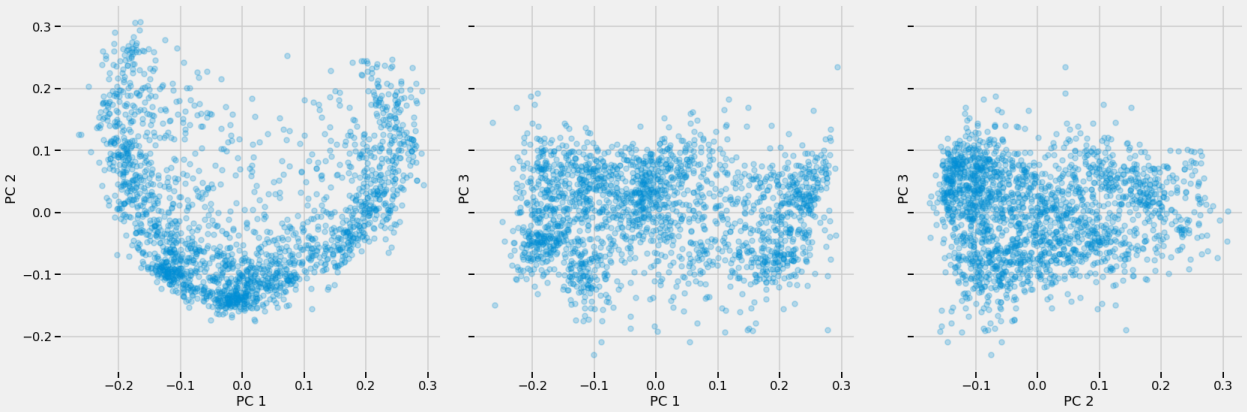


Each piece in this plot is represented by a point in 3-D space. But remember that this location represents ~75% of the information contained in the full tonal pitch-class distribution. In 35-D space each dimension corresponded to the relative frequency of a tonal pitch-class in a piece.

- What do these three dimensions signify?
- How can we interpret them?

Fortunately, we can inspect them individually and try to interpret what we see.

[21]:
```
from itertools import combinations

fig, axes = plt.subplots(1,3, sharey=True, figsize=(24,8))

for k, (i, j) in enumerate(combinations(range(3), 2)):

    axes[k].scatter(X_[:,i], X_[:,j], s=50, alpha=.25, edgecolor=None)
    axes[k].set_xlabel(f"PC {i+1}")
    axes[k].set_ylabel(f"PC {j+1}")
    axes[k].set_aspect("equal")

plt.tight_layout()
# plt.savefig("img/3d_dimension_pairs.png")
# plt.show()
```



Clearly, looking at two principal components at a time shows that there is some latent structure in the data. How can we understand it better?

One way to see whether the pieces are clustered together systematically be coloring them according to some criterion.

As always, many different options are available. For the present purpose we will use the most simple summary of the piece: its most frequent note (which is the *mode* of its pitch-class distribution in statistical terms) and call this note its **tonal center**.

This will also allow to map the tonal pitch-classes on the line of fifths to colors.

[22]:
```
tpc_dists["tonal_center"] = tpc_dists.apply(lambda piece: np.argmax(piece[lof].
↪values) - 15, axis=1)
tpc_dists.sample(10)
```

[22]:

	Fbb	Cbb	Gbb	Dbb	Abb	Ebb	Bbb	Fb	Cb	Gb	\
1990	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.002752	0.004587	
364	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.001026	0.006028	
1857	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.000000	
1045	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.000000	
1246	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.01232	0.020534	0.008214	
214	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.000000	
167	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.000000	
570	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.000000	
586	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.000000	
684	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.00000	0.000000	0.000000	

	Db	Ab	Eb	Bb	F	C	G	\
1990	0.015138	0.088991	0.138991	0.086239	0.117431	0.145413	0.195413	
364	0.020521	0.037194	0.037450	0.016673	0.087598	0.158138	0.225728	
1857	0.000000	0.007370	0.031322	0.103639	0.081069	0.051589	0.111469	
1045	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.004902	
1246	0.131417	0.185832	0.320329	0.091376	0.030801	0.128337	0.070842	
214	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.001401	
167	0.000000	0.000000	0.000000	0.000388	0.028306	0.053121	0.065142	
570	0.000000	0.000000	0.000000	0.004454	0.028211	0.006682	0.092799	
586	0.001506	0.006274	0.041405	0.095609	0.121957	0.107152	0.117440	
684	0.000000	0.000000	0.000000	0.001412	0.072034	0.159605	0.190678	

	D	A	E	B	F#	C#	G#	\
1990	0.094037	0.015596	0.021560	0.055505	0.016972	0.001376	0.000000	
364	0.106579	0.106066	0.094395	0.078107	0.013467	0.002693	0.004232	
1857	0.177338	0.224781	0.087517	0.038692	0.030401	0.037310	0.017503	

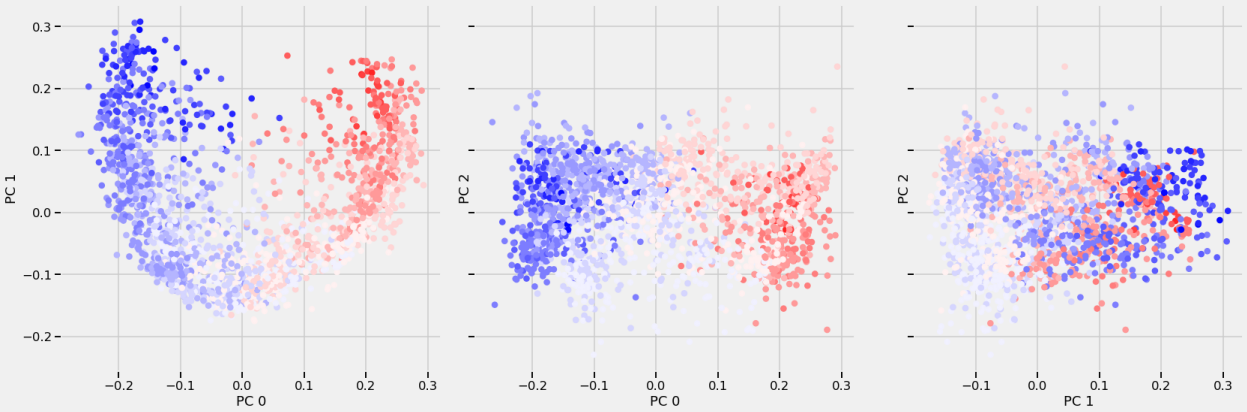
(continues on next page)

(continued from previous page)

```
1045  0.000000  0.000000  0.122549  0.166667  0.151961  0.117647  0.127451
1246  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
214   0.022409  0.082633  0.122549  0.086835  0.147759  0.169468  0.156863
167   0.099651  0.153160  0.186506  0.141528  0.075611  0.089182  0.072896
570   0.171492  0.178916  0.163326  0.086117  0.112843  0.121752  0.029696
586   0.169887  0.146048  0.090088  0.039649  0.021330  0.025094  0.016562
684   0.176554  0.105932  0.149718  0.103107  0.026836  0.011299  0.002825

            D#        A#        E#        B#        F##       C##  G##  D##  \
1990  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.0
364   0.003335  0.000770  0.000000  0.000000  0.000000  0.000000  0.0  0.0
1857  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.0
1045  0.142157  0.102941  0.019608  0.019608  0.009804  0.014706  0.0  0.0
1246  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.0
214   0.111345  0.032913  0.020308  0.037815  0.007003  0.000700  0.0  0.0
167   0.031020  0.001939  0.000775  0.000775  0.000000  0.000000  0.0  0.0
570   0.003712  0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.0
586   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.0
684   0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.0  0.0

      A##  E##  B##  tonal_center
1990  0.0  0.0  0.0             1
364   0.0  0.0  0.0             1
1857  0.0  0.0  0.0             3
1045  0.0  0.0  0.0             5
1246  0.0  0.0  0.0            -3
214   0.0  0.0  0.0             7
167   0.0  0.0  0.0             4
570   0.0  0.0  0.0             3
586   0.0  0.0  0.0             2
684   0.0  0.0  0.0             1
```

[23]:
```
from matplotlib import cm
from matplotlib.colors import Normalize

#normalize item number values to colormap
norm = Normalize(vmin=-15, vmax=20)

# cs = [ cm.seismic(norm(c)) for c in data["tonal_center"]]
cs = [ cm.seismic(norm(c)) for c in tpc_dists["tonal_center"]]
```

[24]:
```
from itertools import combinations

fig, axes = plt.subplots(1,3, sharey=True, figsize=(24,8))

for k, (i, j) in enumerate(combinations(range(3), 2)):

    axes[k].scatter(X_[:,i], X_[:,j], s=50, c=[ np.abs(c) for c in cs],␣
↪edgecolor=None)
    axes[k].set_xlabel(f"PC {i}")
    axes[k].set_ylabel(f"PC {j}")
    axes[k].set_aspect("equal")

plt.tight_layout()
# plt.savefig("img/3d_dimension_pairs_colored.png")
# plt.show()
```



8.5 Historical development of tonality

The line of fifths is an important underlying structure for pitch-class distributions in tonal compositions

But we have treated all pieces in our dataset as synchronic and have not yet taken their historical location into account.

Let's assume the pitch-class content of a piece spreads on the line of fifths from F to A♯. This means, its range on the line of fifths is $10 - (-1) = 11$. The piece covers eleven consecutive fifths on the lof.

We can generalize this calculation and write a function that calculates the range for each piece in the dataset.

[25]:
```
def lof_range(piece):
    l = [i for i, v in enumerate(piece) if v!=0]
    return max(l) - min(l)
```

[26]:
```
data["lof_range"] = data.loc[:, lof].apply(lof_range, axis=1) # create a new column
data.sample(20)
```

[26]:

	composer	composer_first	work_group	\
992	Agricola	Alexander	Missa Malheur me bat	
1772	Corelli	Arcangelo	12 Trio Sonatas	
1199	Scarlatti	Domenico	Sonata	
1361	Scriabin	Alexander	Préludes	
1490	Tchaikovsky	Pyotr	The Seasons	
387	Chopin	Frédéric	Préludes	
290	Alkan	Charles Valentin	Préludes	
478	Bach	Johann Sebastian	Inventions and Sinfonias	
1947	Mozart	Wolfgang Amadeus	Sonaten	
601	Couperin	François	Troisième livre de pièces de Clavecin	
974	Ockeghem	JeanDe	Missa Fors Seulement	
1430	Victoria	TomasLuisde	NaN	
528	Brahms	Johannes	8 Klavierstücke	
1765	Corelli	Arcangelo	12 Trio Sonatas	
1300	Schumann	Clara	Sechs Lieder	
204	Liszt	Franz	12 Transcendental Etudes	
1858	Grieg	Edvard	Lyrical Pieces	
468	Bach	Johann Sebastian	Inventions and Sinfonias	
439	Victoria	TomasLuisde	NaN	
441	Alkan	Charles Valentin	NaN	

	work_catalogue	opus	no	mov	title	\

(continues on next page)

(continued from previous page)

```
992             NaN  NaN  NaN  NaN                              Gloria
1772           Op.     4    1  3.0                                 NaN
1199              K   64  NaN  NaN                                 NaN
1361            Op.   13    2  NaN                           6 Preludes
1490            Op.  37a    3  NaN            March: Song of the Lark
387             Op.   28    6  NaN                          24 Préludes
290             Op.   31   13  NaN                                 NaN
478             BWV  789  NaN  NaN                                 NaN
1947             KV  283    5  3.0                                 NaN
601             NaN  NaN   18  4.0                       Le petit rien
974             NaN  NaN  NaN  NaN                               Kyrie
1430            NaN  NaN  NaN  NaN                      Sepulto Domino
528             Op.   76    2  NaN                           Capriccio
1765           Op.     3    8  4.0                                 NaN
1300            Op.   23    5  NaN  Das ist der Tag, der klingen mag
204              S.  139    3  NaN                             Paysage
1858            Op.   65    6  NaN                         Bryllupsdag
468             BWV  779  NaN  NaN                                 NaN
439             NaN  NaN  NaN  NaN          Aleph. Quomodo obscuratum
441             Op.   25  NaN  NaN                            Alleluia

      composition  publication source  display_year  Fbb  Cbb  Gbb  Dbb  Abb  \
992           NaN       1506.0  ELVIS        1506.0    0    0    0    0    0
1772          NaN       1694.0  CCARH        1694.0    0    0    0    0    0
1199          NaN          NaN     MS        1721.0    0    0    0    0    0
1361       1895.0          NaN   DCML        1895.0    0    0    0    0    0
1490       1876.0          NaN     MS        1876.0    0    0    0    0    0
387        1839.0          NaN     MS        1839.0    0    0    0    0    0
290        1846.0          NaN     MS        1846.0    0    0    0    0    1
478           NaN       1723.0     MS        1723.0    0    0    0    0    0
1947       1774.0          NaN  CCARH        1774.0    0    0    0    0    0
601           NaN       1722.0     MS        1722.0    0    0    0    0    0
974        1497.0          NaN  ELVIS        1497.0    0    0    0    0    0
1430          NaN       1585.0  ELVIS        1585.0    0    0    0    0    0
528        1878.0          NaN   DCML        1878.0    0    0    0    0    0
1765          NaN       1689.0  CCARH        1689.0    0    0    0    0    0
1300       1853.0          NaN   OSLC        1853.0    0    0    0    0    0
204        1851.0          NaN     MS        1851.0    0    0    0    0    0
1858       1896.0       1897.0   DCML        1896.0    0    0    0    0    0
468           NaN       1723.0     MS        1723.0    0    0    0    0    0
439           NaN       1585.0  ELVIS        1585.0    0    0    0    0    0
441           NaN       1844.0     MS        1844.0    0    0    0    0    0

      Ebb  Bbb  Fb   Cb   Gb   Db   Ab   Eb   Bb    F    C    G    D    A  \
992     0    0   0    0    0    0    0    0    3  218  386  327  274  333
1772    0    0   0    0    0    0    0    0    0   28   50   47   27   39
1199    0    0   0    0    0    0    0    0   30   62   32   60   89  129
1361    0    0   0    0    0    0    0    8    6   78   61   27   40   91
1490    0    0   0    0    3    0    0   24  107   17   48  126  158   71
387     0    0   0    0    0    0    0    0    0    1   20   41   80    4
290     7   22   3  108  201  263  110   69  228  150   10    1   20    6
478     0    0   0    0    0    0    0    0    0    1   23   82   77  103
1947    0    0   0    0    0    0    0    8    9   29  183  367  317  245
601     0    0   0    0    0    0    0    0    0    0    1   29   55   56
974     0    0   0    0    0    0    0    0   25  131  124  106  158  167
1430    0    0   0    0   13    1    0   19   56   30   47   79  108   49
528     0    0   0    0    1    2    2    8   18   44   53  147  144  116
```

(continues on next page)

(continued from previous page)

```
1765    0    0    0    0    0    0    0    0    5  163  156  207  106  170
1300    0    0    0    0    0    0    0    0    0    0    0   29  126  174
204    10    2    3   10   36  109   66   72  168  519  243  124  173  279
1858    0    0    0   12   52   68  236  256  104  112  243  415  582  555
468     0    0    0    0    0    0    0   17   68   98   92   77   82   81
439     0    0    0    0    0   11   33    0    2   82  117   82   73  195
441     0    0    0    0    0  140    0   17  231  420  309  269  166  552

         E    B   F#   C#  G#  D#  A#  E#  B#  F##  C##  G##  D##  A##  E##  \
992    351  298    5    3   7   0   0   0   0    0    0    0    0    0    0
1772    53   38    4    2   7   2   0   0   0    0    0    0    0    0    0
1199    72   17    3   22  11   0   0   0   0    0    0    0    0    0    0
1361    70   46    4    3  12  16   4   0   0    0    0    0    0    0    0
1490     1    4   52   13   2   0   0   0   0    0    0    0    0    0    0
387     28  106   66   33   3   0  22   2   0    1    0    0    0    0    0
290      1    0    0    0   0   0   0   0   0    0    0    0    0    0    0
478     98  101  109   68  23  13   8   5   0    0    0    0    0    0    0
1947   184  269  199   70  33  32  38   1   0    0    0    0    0    0    0
601     51   28   44   29   4   0   0   0   0    0    0    0    0    0    0
974    104   60    0    5   1   0   0   0   0    0    0    0    0    0    0
1430    15    9    0    0   0   0   0   0   0    0    0    0    0    0    0
528    197  233  307  203  88  83  95  50   9    0    2    0    1    0    0
1765   195  127   12    0   6   7   0   0   0    0    0    0    0    0    0
1300    93   79  106   58  30   7   2   0   0    0    0    0    0    0    0
204    115   42   13   20  14   1   0   0   0    0    0    0    0    0    0
1858   254  346  315  110  33  13   7   1   1    0    0    0    0    0    0
468     56   17    3    7   0   0   0   0   0    0    0    0    0    0    0
439    149   77   15    0   0   0   0   0   0    0    0    0    0    0    0
441    235   82   35    9  35  13   0   0   0    0    0    0    0    0    0

       B##  lof_range
992      0         10
1772     0         10
1199     0         10
1361     0         13
1490     0         14
387      0         14
290      0         15
478      0         12
1947     0         14
601      0          8
974      0         10
1430     0         11
528      0         22
1765     0         11
1300     0          9
204      0         19
1858     0         19
468      0         10
439      0         11
441      0         14
```

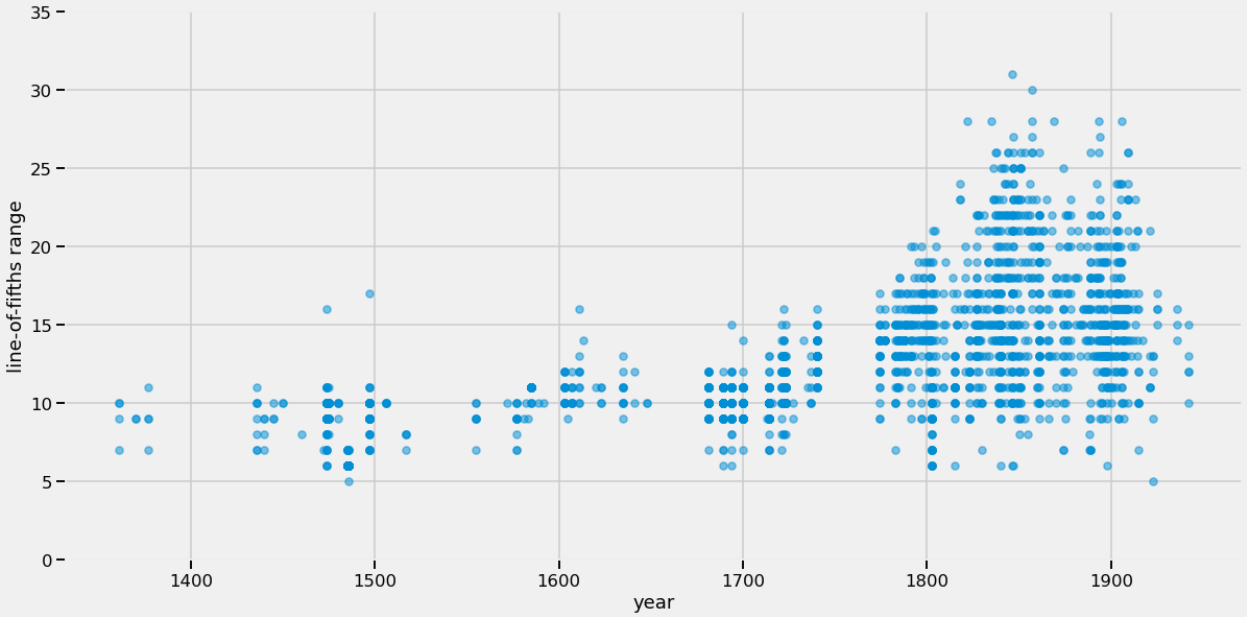
This allows us now to take the `display_year` (composition or publication) and `lof_range` (range on the line of fifths) features to observe historical changes.

[30]:
```
fig, ax = plt.subplots(figsize=(18,9))
```

(continues on next page)

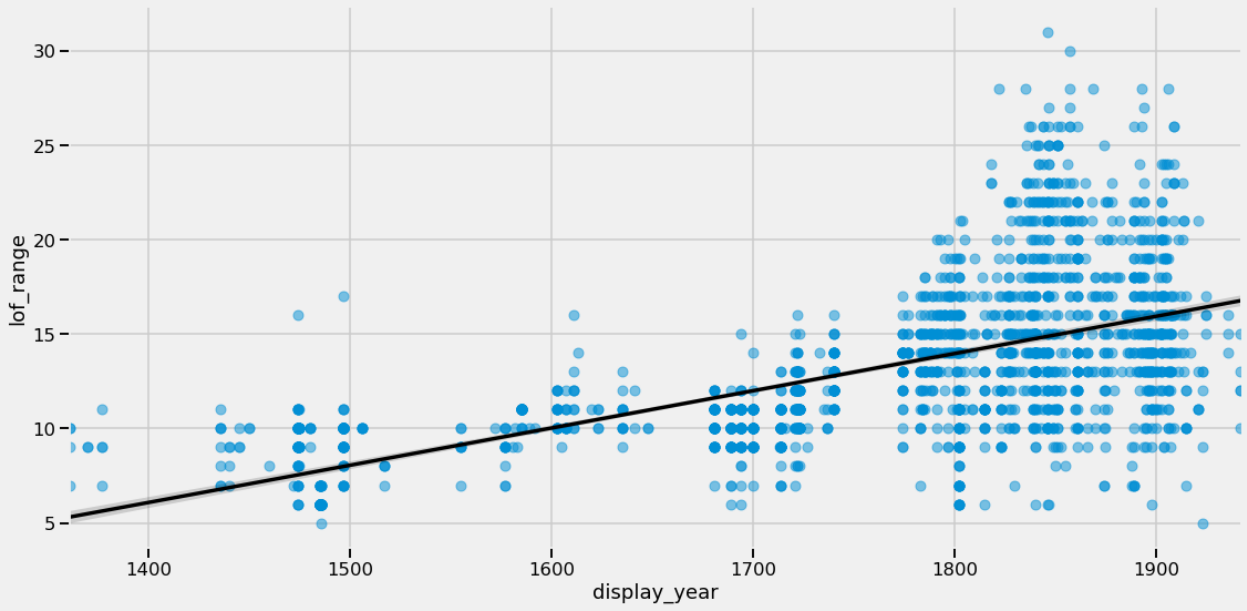
(continued from previous page)

```
ax.scatter(data["display_year"].values, data["lof_range"].values, alpha=.5, s=50)
ax.set_ylim(0,35)
ax.set_xlabel("year")
ax.set_ylabel("line-of-fifths range");
# plt.savefig("img/hist_scatter.png");
```



We could try to fit a line to this data to see whether there is a trend (kinda obvious here).

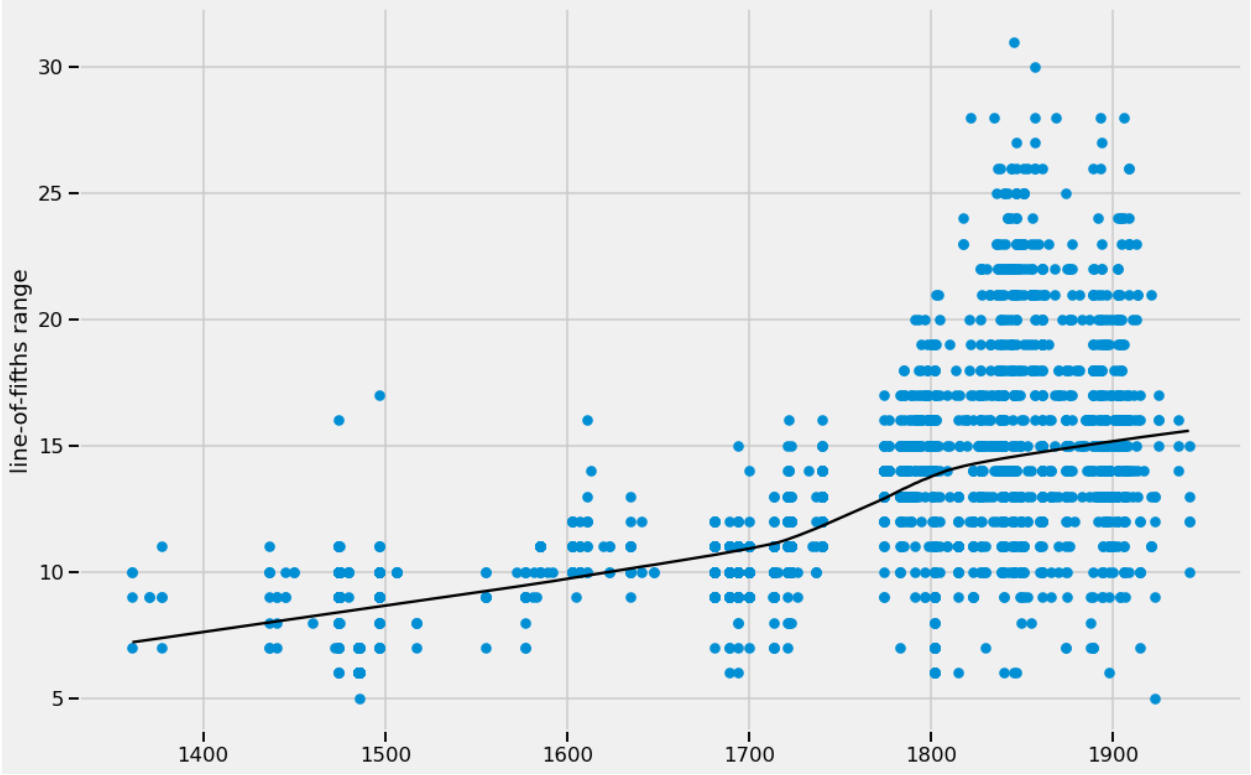
[32]:
```
g = sns.lmplot(
    data=data,
    x="display_year",
    y="lof_range",
    line_kws={"color":"k"},
    scatter_kws={"alpha":.5},
#     lowess=True,
    height=8,
    aspect=2
);
# g.savefig("img/hist_scatter_line.png");
```



But actually, this is not the best idea. Why should any historical process be linear? More complex models might make more sense.

A more versatile technique is *Locally Weighted Scatterplot Smoothing* (LOWESS) that locally fits a polynomial. Using this method, we see that a non-linear process is displayed.

[34]:
```
from statsmodels.nonparametric.smoothers_lowess import lowess

x = data.display_year
y = data.lof_range
l = lowess(y,x)

fig, ax = plt.subplots(figsize=(15,10))

ax.scatter(x,y, s=50)
ax.plot(l[:,0], l[:,1], c="k")
ax.set_ylabel("line-of-fifths range");
# plt.savefig("img/hist_scatter_lowess.png")
# plt.show()
```



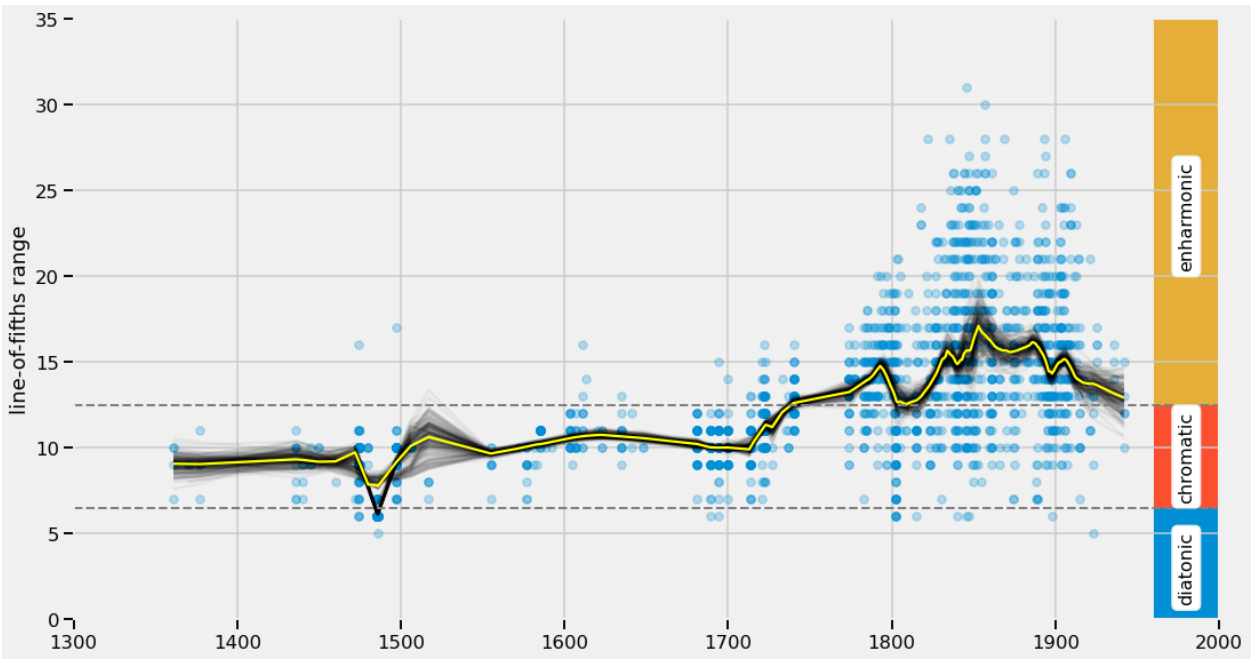
8.6 If there is time: some more advanced stuff

[35]:
```
B = 200
delta = 1/10

fig, ax = plt.subplots(figsize=(16,9))

x = data.display_year
y = data.lof_range
l = lowess(y,x, frac=delta)

ax.scatter(x,y, s=50, alpha=.25)

for _ in range(B):
    resampled = data.sample(data.shape[0], replace=True)

    xx = resampled.display_year
    yy = resampled.lof_range
    ll = lowess(yy,xx, frac=delta)

    ax.plot(ll[:,0], ll[:,1], c="k", alpha=.05)

ax.plot(l[:,0], l[:,1], c="yellow")

## REGIONS
from matplotlib.patches import Rectangle
```

(continues on next page)

(continued from previous page)

```
text_kws = {
    "rotation" : 90,
    "fontsize" : 16,
    "bbox" : dict(
        facecolor="white",
        boxstyle="round"
    ),
    "horizontalalignment" : "center",
    "verticalalignment" : "center"
}

rect_props = {
    "width" : 40,
    "zorder" : -1,
    "alpha" : 1.
}

stylecolors = plt.rcParams["axes.prop_cycle"].by_key()["color"]

ax.text(1980, 3, "diatonic", **text_kws)
ax.axhline(6.5, c="gray", linestyle="--", lw=2) # dia / chrom.
ax.add_patch(Rectangle((1960,0), height=6.5, facecolor=stylecolors[0], **rect_props))

ax.text(1980, 9.5, "chromatic", **text_kws)
ax.axhline(12.5, c="gray", linestyle="--", lw=2) # chr. / enh.
ax.add_patch(Rectangle((1960,6.5), height=6, facecolor=stylecolors[1], **rect_props))

ax.text(1980, 23.5, "enharmonic", **text_kws)
ax.add_patch(Rectangle((1960,12.5), height=28, facecolor=stylecolors[2], **rect_
↪props))

ax.set_ylim(0,35)
ax.set_xlim(1300,2000)

ax.set_ylabel("line-of-fifths range");
# plt.savefig("img/final.png", dpi=300)
# plt.show()
```



Usung bootstrap sampling we achieve an estimation of the local varience of the data and thus of the diversity in the note usage of the musical pieces.

We also can distinguish three regions in terms of line-of-fifth range: diatonic, chromatic, and enharmonic.

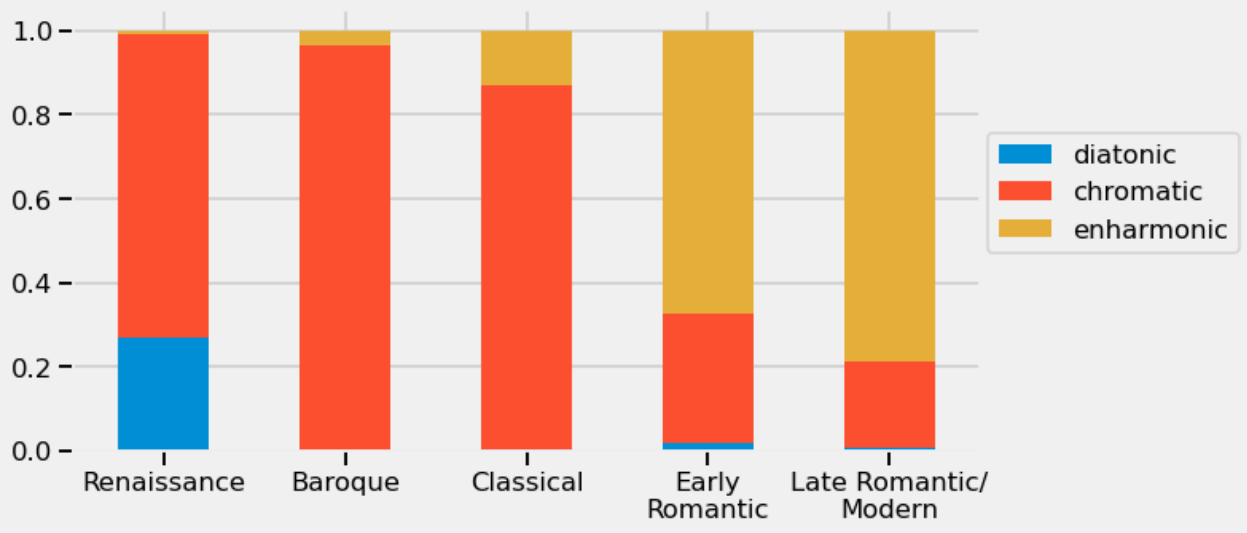
Grouping the data together in these three regions, we see a clear change from diatonic and chromatic to chromatic and enharmonic pieces over the course of history.

[36]:
```
epochs = {
    "Renaissance" : [1300, 1549],
    "Baroque" : [1550, 1649],
    "Classical" : [1650, 1749],
    "Early\nRomantic" : [1750, 1819],
    "Late Romantic/\nModern" : [1820, 2000]
}

strata = [
    "diatonic",
    "chromatic",
    "enharmonic"
]

widths = data[["display_year", "lof_range"]].sort_values(by="display_year").reset_
↪index(drop=True)

df = pd.concat(
    [
        widths[
            (widths.display_year >= epochs[e][0]) & (widths.display_year <=␣
↪epochs[e][1])
        ]["lof_range"].value_counts(normalize=True).sort_index().groupby(
            lambda x: strata[0] if x <= 6 else strata[1] if x <= 12 else strata[2]
        ).sum() for e in epochs
    ], axis=1, sort=True
```

(continues on next page)

(continued from previous page)

```
)

df.columns = epochs.keys()
df = df.reindex(strata)
df.T.plot(kind="bar", stacked=True, figsize=(12,5))
# plt.title("Epochs")
plt.legend(bbox_to_anchor=(1.3,0.75))
plt.gca().set_xticklabels(epochs.keys(), rotation="horizontal")
plt.tight_layout()
# plt.savefig("img/epochs_regions.png")
plt.show()
```



- Renaissance: largest diatonic proportion overall but mostly chromatic
- Baroque: alost completely chromatic
- Classical: enharmonic proportion increases -> more distant modulations
- This trend continues through the Romantic eras

8.7 Summary

1. We have analyzed a very specific aspect of Western classical music.
2. We have used a large(-ish) corpus to answer our research question.
3. We have operationalized musical pieces as vectors that represent distributions of tonal pitch-classes.
4. We have used the dimensionality-reduction technique Principal Component Analysis (PCA) in order to visually inspect the distribution of the data in 2 and 3 dimensions.
5. We have used music-theoretical domain knowledge to find meaningful structure in this space.
6. We have seen that pieces are largely distributed along the line of fifths.
7. We have used Locally Weighted Scatterplot Smoothing (LOWESS) to estimate the variance in this historical process.
8. We have seen that, historically, composers explore ever larger regions on this line and that the variance also increases.

[ ]: