

---

# **Introduction to Musical Corpus Studies**

***Release 0.0.1***

**Fabian C. Moss**

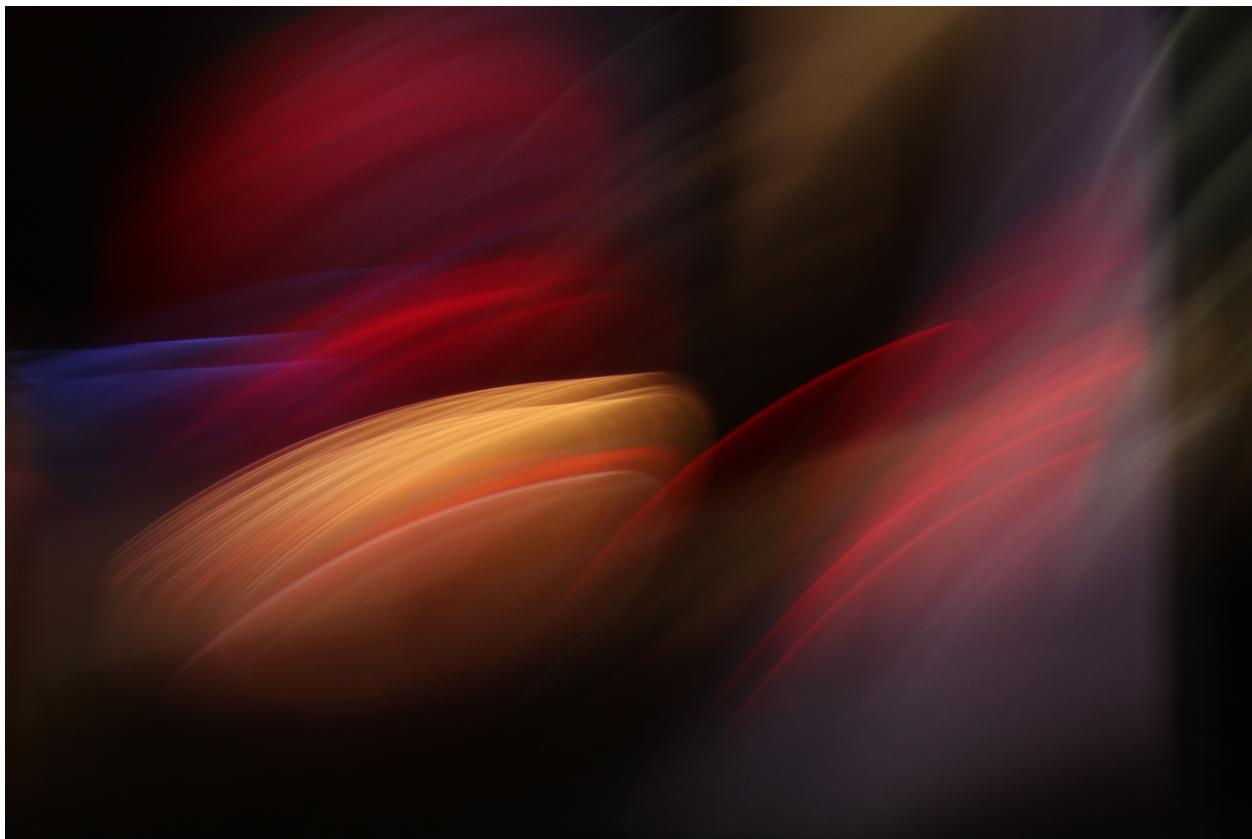
**Nov 14, 2020**



# CONTENT

<b>1 Organization</b>	<b>3</b>
1.1 About this course . . . . .	3
1.2 Overview . . . . .	3
1.3 Credits . . . . .	4
1.4 Deliverables and learning objectives . . . . .	4
<b>2 Introduction and Background</b>	<b>7</b>
<b>3 Melodies in Folk Songs</b>	<b>9</b>
3.1 The <i>Essen Folksong Collection</i> . . . . .	9
3.2 Comparing songs . . . . .	14
3.3 Computational analysis . . . . .	15
3.4 The melodic arc . . . . .	19
3.5 Intervals . . . . .	20
<b>4 Solos in the Weimar Jazz Database</b>	<b>23</b>
4.1 Melodic arc? . . . . .	27
4.2 Pitch vs loudness . . . . .	31
4.3 The “rain cloud” of Jazz solos . . . . .	32
4.4 Comparing performers . . . . .	33





**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-disciplines (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.



**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 actually 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 cadences 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	<i>Lunch Break</i>
5		13:00-14:20 Uhr	Group work
6		14:40-16:00 Uhr	Harmony I: Beethoven's string quartets
7	Fr., 11.12.2020	10:00-11:20 Uhr	Harmony II: Cadences in Renaissance polyphony (guest: <a href="#">Richard Freedman</a> )
8		11:40-13:00 Uhr	Harmony III & Form: Brazilian Choro / Pop charts (Billboard 100)
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	<i>Lunch Break</i>
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 of 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](mailto: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](#)

---

## 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")

[ ]: ## 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 maintained by 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`.

```
[2]: # 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`.

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

```
[3]: 31
```

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

```
[4]: corpora
```

```
[4]: [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 triple 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 0x235ff1e0730>
```

It returns a somewhat cryptic statement that says that the variable counts a `music21.stream.Score` object. Understanding the internal organization of `music21` goes beyond this class. For us, it is sufficient 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()
```

## Speed the Plough

Trad.

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

```
[14]: corpora[0]
```

```
[14]: 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!

```
[15]: 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.

```
[16]: first_corpus
```

```
[16]: <music21.stream.Opus 0x235ff39ad60>
```

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. If 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 7st piece? Remember that Python starts counting at 0.

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

## Die plappernden Junggesellen



14



A A B A'

```
[18]: 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).

```
[19]: 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.

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

```
[20]: 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`.

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

It is easy to display these songs now:

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

Die plappernden Junggesellen



14



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

# Shengsi liangxianglian

Musical score for piano, page 12, measures 12-13. The score consists of two staves. The top staff uses a treble clef and a key signature of one flat (B-flat). It features a mix of eighth and sixteenth-note patterns, including grace notes and slurs. The bottom staff uses a treble clef and a key signature of one flat. Measures 12 and 13 begin with quarter notes, followed by eighth-note patterns. The music concludes with a final measure ending on a half note.

```
[24]: 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 will 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:

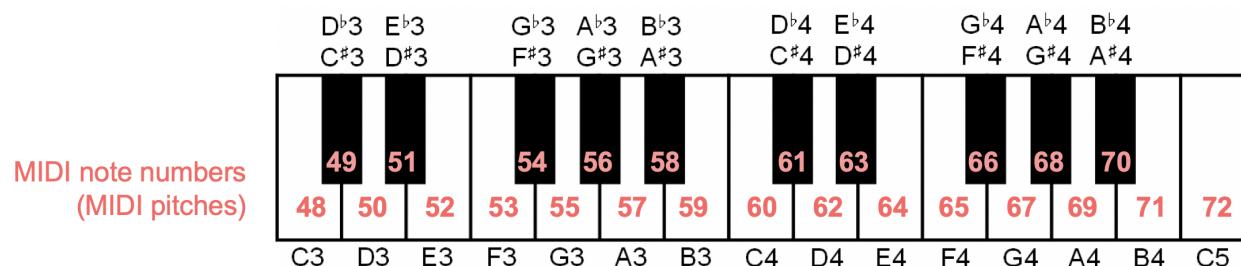


Image from [https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S2\\_MIDI.html](https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S2_MIDI.html).

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.

```
[25]: 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:

[27]: notelist(german\_song) [:8]

	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.

```
[1]: def plot_melodic_profile(notelist, ax=None, c=None, mean=False, Z=False,
    ~sections=False, standardized=False):

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

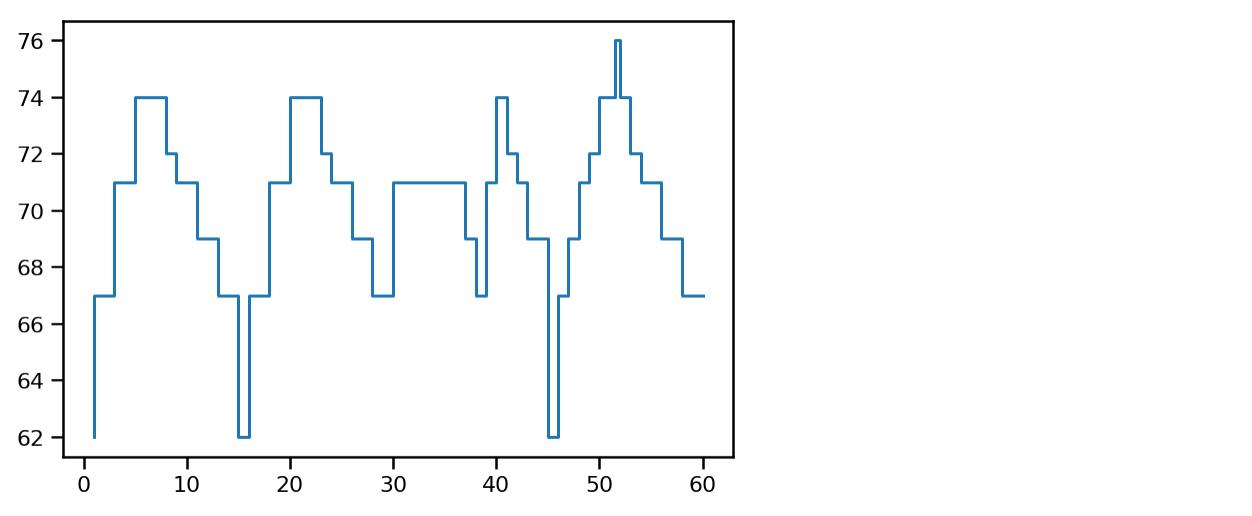
    if standardized:
        notelist = standardized(notelist)
        x = notelist["Rel. Onset"]
        y = notelist["Rel. MIDI Pitch"]
    else:
        x = notelist["Onset"]
        y = notelist["MIDI Pitch"]

    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="--")
```

[29]: plot\_melodic\_profile(notelist(german\_song))

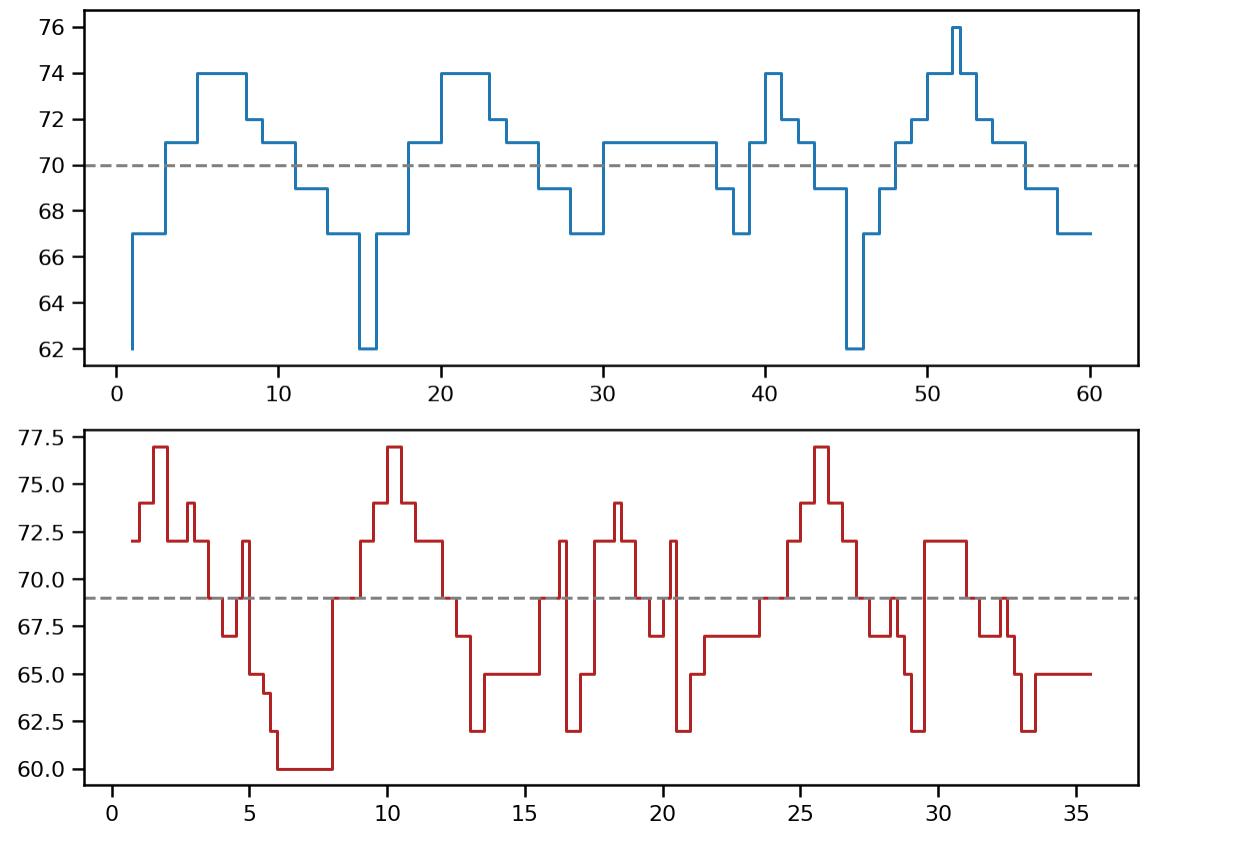


Likewise, we can as easily plot the melodic contour of the Chinese song (we will use a different color).

```
[30]: 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()`.

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

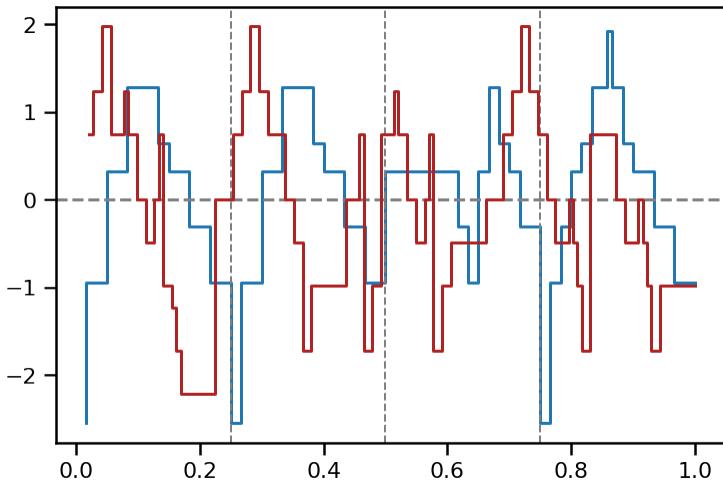
    notelist["Rel. MIDI Pitch"] = (notelist["MIDI Pitch"] - notelist["MIDI Pitch"].mean()) / notelist["MIDI Pitch"].std()
    notelist["Rel. Duration"] = notelist["Duration"] / notelist["Duration"].sum()
    notelist["Rel. Onset"] = notelist["Onset"] / notelist["Onset"].max()

    return notelist
```

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

	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

```
[34]: 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.

```
[ ]: 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)

[ ]: big_df

[ ]: big_df.to_csv("data/big_df.csv")

[ ]: big_df.sample(10)
```

## 3.4 The melodic arc

```
[ ]: %%time

fig, ax = plt.subplots(figsize=(5,5))

grouped = big_df.groupby("Song ID")

for i, g in grouped:
    x = g["Rel. Onset"]
    y = g["Rel. MIDI Pitch"]
    ax.plot(x,y, lw=.5, c="tab:red", alpha=1/300)

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)
ax.plot(big_z[:,0], big_z[:,1], c="black", lw=2)

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()
```

## 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*

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

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

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

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 perfectly balanced. Note that -2 (two descending semitones, or one descending whole tone) is the most frequent interval.

```
[ ]: 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)
```

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

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

```
[ ]: 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
```

```
[ ]: ints_df
```

```
[ ]: 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.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?

```
[ ]: 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
```

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

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

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

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

```
[ ]: %%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
```

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

```
[ ]: 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()
```



## SOLOS IN THE WEIMAR JAZZ DATABASE

### Disclaimer: I am not the expert here!

In this session, we will have a look at 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](#)).

```
[85]: 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/wjazzd.db`. We use the `sqlite3` library to connect to this database.

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

```
[3]: conn
```

```
[3]: <sqlite3.Connection at 0x17f98cc85d0>
```

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:

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

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

```
[18]: solos_meta.shape
```

```
[18]: (456, 17)
```

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

[8]:	solos_meta.sample(10)																																																																													
[8]:	<table border="1"> <thead> <tr> <th>melid</th><th>trackid</th><th>compid</th><th>recordid</th><th>performer</th><th>title</th><th>\</th></tr> </thead> <tbody> <tr><td>170</td><td>171</td><td>150</td><td>135</td><td>74 George Coleman</td><td>Maiden Voyage</td><td></td></tr> <tr><td>393</td><td>394</td><td>171</td><td>155</td><td>86 Sonny Stitt</td><td>Elora</td><td></td></tr> <tr><td>125</td><td>126</td><td>117</td><td>108</td><td>59 Dexter Gordon</td><td>The Rainbow People</td><td></td></tr> <tr><td>264</td><td>265</td><td>228</td><td>206</td><td>108 Kid Ory</td><td>Who's it</td><td></td></tr> <tr><td>314</td><td>315</td><td>259</td><td>235</td><td>140 Miles Davis</td><td>E.S.P.</td><td></td></tr> <tr><td>440</td><td>441</td><td>335</td><td>295</td><td>180 Woody Shaw</td><td>Rosewood</td><td></td></tr> <tr><td>231</td><td>232</td><td>205</td><td>183</td><td>106 John Coltrane</td><td>Soultrane</td><td></td></tr> <tr><td>198</td><td>199</td><td>176</td><td>160</td><td>88 Joe Henderson</td><td>In 'n Out</td><td></td></tr> <tr><td>333</td><td>334</td><td>270</td><td>244</td><td>18 Nat Adderley</td><td>Bohemia After Dark</td><td></td></tr> <tr><td>181</td><td>182</td><td>18</td><td>18</td><td>8 Harry Edison</td><td>Did You Call Her Today</td><td></td></tr> </tbody> </table>	melid	trackid	compid	recordid	performer	title	\	170	171	150	135	74 George Coleman	Maiden Voyage		393	394	171	155	86 Sonny Stitt	Elora		125	126	117	108	59 Dexter Gordon	The Rainbow People		264	265	228	206	108 Kid Ory	Who's it		314	315	259	235	140 Miles Davis	E.S.P.		440	441	335	295	180 Woody Shaw	Rosewood		231	232	205	183	106 John Coltrane	Soultrane		198	199	176	160	88 Joe Henderson	In 'n Out		333	334	270	244	18 Nat Adderley	Bohemia After Dark		181	182	18	18	8 Harry Edison	Did You Call Her Today	
melid	trackid	compid	recordid	performer	title	\																																																																								
170	171	150	135	74 George Coleman	Maiden Voyage																																																																									
393	394	171	155	86 Sonny Stitt	Elora																																																																									
125	126	117	108	59 Dexter Gordon	The Rainbow People																																																																									
264	265	228	206	108 Kid Ory	Who's it																																																																									
314	315	259	235	140 Miles Davis	E.S.P.																																																																									
440	441	335	295	180 Woody Shaw	Rosewood																																																																									
231	232	205	183	106 John Coltrane	Soultrane																																																																									
198	199	176	160	88 Joe Henderson	In 'n Out																																																																									
333	334	270	244	18 Nat Adderley	Bohemia After Dark																																																																									
181	182	18	18	8 Harry Edison	Did You Call Her Today																																																																									
[8]:	<table border="1"> <thead> <tr> <th>titleaddon</th><th>solo(part)</th><th>instrument</th><th>style</th><th>avgtempo</th><th>tempoclass</th><th>\</th></tr> </thead> <tbody> <tr><td>170</td><td>1</td><td>ts</td><td>POSTBOP</td><td>125.4</td><td>MEDIUM</td><td></td></tr> <tr><td>393</td><td>1</td><td>ts</td><td>BEBOP</td><td>192.7</td><td>UP</td><td></td></tr> <tr><td>125</td><td>1</td><td>ts</td><td>HARDBOP</td><td>120.0</td><td>MEDIUM</td><td></td></tr> <tr><td>264</td><td>1</td><td>tb</td><td>TRADITIONAL</td><td>211.6</td><td>UP</td><td></td></tr> <tr><td>314</td><td>1</td><td>tp</td><td>POSTBOP</td><td>291.4</td><td>UP</td><td></td></tr> <tr><td>440</td><td>1</td><td>tp</td><td>POSTBOP</td><td>171.7</td><td>MEDIUM UP</td><td></td></tr> <tr><td>231</td><td>1</td><td>ts</td><td>HARDBOP</td><td>82.1</td><td>MEDIUM SLOW</td><td></td></tr> <tr><td>198</td><td>2</td><td>ts</td><td>POSTBOP</td><td>299.5</td><td>UP</td><td></td></tr> <tr><td>333</td><td>1</td><td>tp</td><td>HARDBOP</td><td>295.4</td><td>UP</td><td></td></tr> <tr><td>181</td><td>1</td><td>tp</td><td>SWING</td><td>118.5</td><td>MEDIUM</td><td></td></tr> </tbody> </table>	titleaddon	solo(part)	instrument	style	avgtempo	tempoclass	\	170	1	ts	POSTBOP	125.4	MEDIUM		393	1	ts	BEBOP	192.7	UP		125	1	ts	HARDBOP	120.0	MEDIUM		264	1	tb	TRADITIONAL	211.6	UP		314	1	tp	POSTBOP	291.4	UP		440	1	tp	POSTBOP	171.7	MEDIUM UP		231	1	ts	HARDBOP	82.1	MEDIUM SLOW		198	2	ts	POSTBOP	299.5	UP		333	1	tp	HARDBOP	295.4	UP		181	1	tp	SWING	118.5	MEDIUM	
titleaddon	solo(part)	instrument	style	avgtempo	tempoclass	\																																																																								
170	1	ts	POSTBOP	125.4	MEDIUM																																																																									
393	1	ts	BEBOP	192.7	UP																																																																									
125	1	ts	HARDBOP	120.0	MEDIUM																																																																									
264	1	tb	TRADITIONAL	211.6	UP																																																																									
314	1	tp	POSTBOP	291.4	UP																																																																									
440	1	tp	POSTBOP	171.7	MEDIUM UP																																																																									
231	1	ts	HARDBOP	82.1	MEDIUM SLOW																																																																									
198	2	ts	POSTBOP	299.5	UP																																																																									
333	1	tp	HARDBOP	295.4	UP																																																																									
181	1	tp	SWING	118.5	MEDIUM																																																																									
[8]:	<table border="1"> <thead> <tr> <th>rhythmfeel</th><th>key</th><th>signature</th><th>\</th></tr> </thead> <tbody> <tr><td>170</td><td>SWING</td><td>4/4</td><td></td></tr> <tr><td>393</td><td>SWING</td><td>Bb-maj</td><td>4/4</td></tr> <tr><td>125</td><td>SWING</td><td>Eb-maj</td><td>4/4</td></tr> <tr><td>264</td><td>TWOBEAT</td><td>C-maj</td><td>4/4</td></tr> <tr><td>314</td><td>SWING</td><td>F-maj</td><td>4/4</td></tr> <tr><td>440</td><td>LATIN/FUNK</td><td></td><td>4/4</td></tr> <tr><td>231</td><td>SWING</td><td>Eb-maj</td><td>4/4</td></tr> <tr><td>198</td><td>SWING</td><td>Eb-maj</td><td>4/4</td></tr> <tr><td>333</td><td>SWING</td><td>G-min</td><td>4/4</td></tr> <tr><td>181</td><td>SWING</td><td>Ab-maj</td><td>4/4</td></tr> </tbody> </table>	rhythmfeel	key	signature	\	170	SWING	4/4		393	SWING	Bb-maj	4/4	125	SWING	Eb-maj	4/4	264	TWOBEAT	C-maj	4/4	314	SWING	F-maj	4/4	440	LATIN/FUNK		4/4	231	SWING	Eb-maj	4/4	198	SWING	Eb-maj	4/4	333	SWING	G-min	4/4	181	SWING	Ab-maj	4/4																																	
rhythmfeel	key	signature	\																																																																											
170	SWING	4/4																																																																												
393	SWING	Bb-maj	4/4																																																																											
125	SWING	Eb-maj	4/4																																																																											
264	TWOBEAT	C-maj	4/4																																																																											
314	SWING	F-maj	4/4																																																																											
440	LATIN/FUNK		4/4																																																																											
231	SWING	Eb-maj	4/4																																																																											
198	SWING	Eb-maj	4/4																																																																											
333	SWING	G-min	4/4																																																																											
181	SWING	Ab-maj	4/4																																																																											
[8]:	<table border="1"> <thead> <tr> <th></th><th>chord_changes</th><th>chorus_count</th></tr> </thead> <tbody> <tr><td>170 A1:   Dsus7  Dsus7  Dsus7  Dsus7  Fsus...</td><td>1</td><td></td></tr> <tr><td>393 A1:   C-7  F7  Bbj7 Ebj7  D-7 Db-7  C-7 ...</td><td>1</td><td></td></tr> <tr><td>125 A1:   Eb F-7  G-7 F-7  G-7 F-7  F#-7 B7  F-7 ...</td><td>3</td><td></td></tr> <tr><td>264 A1:   C  A7  D7  D7  G7  G7  C  ...  </td><td>1</td><td></td></tr> <tr><td>314 A1:   E7alt  E7alt  Fj7  Fj7  E7alt  ...  </td><td>6</td><td></td></tr> <tr><td>440 A1:   G-79 F-79  G-79 F-79  C-79 Bb-79  Gb79  ...</td><td>1</td><td></td></tr> <tr><td>231 A1:   Ebj7  Gm7b5 C7  F-7  Db7  Ab-7 Db7...  </td><td>1</td><td></td></tr> <tr><td>198 A1:   Fsus7913  Fsus7913  Fsus7913  Gbsu...  </td><td>8</td><td></td></tr> <tr><td>333 A1:   D79b  D79b   \nA2:   G-7  Am7b5 D7...  </td><td>3</td><td></td></tr> <tr><td>181 A1:   Bb7  Eb7  Ab6  Ab6  Eb-7  Ab7  ...  </td><td>2</td><td></td></tr> </tbody> </table>		chord_changes	chorus_count	170 A1:   Dsus7  Dsus7  Dsus7  Dsus7  Fsus...	1		393 A1:   C-7  F7  Bbj7 Ebj7  D-7 Db-7  C-7 ...	1		125 A1:   Eb F-7  G-7 F-7  G-7 F-7  F#-7 B7  F-7 ...	3		264 A1:   C  A7  D7  D7  G7  G7  C  ...	1		314 A1:   E7alt  E7alt  Fj7  Fj7  E7alt  ...	6		440 A1:   G-79 F-79  G-79 F-79  C-79 Bb-79  Gb79  ...	1		231 A1:   Ebj7  Gm7b5 C7  F-7  Db7  Ab-7 Db7...	1		198 A1:   Fsus7913  Fsus7913  Fsus7913  Gbsu...	8		333 A1:   D79b  D79b   \nA2:   G-7  Am7b5 D7...	3		181 A1:   Bb7  Eb7  Ab6  Ab6  Eb-7  Ab7  ...	2																																													
	chord_changes	chorus_count																																																																												
170 A1:   Dsus7  Dsus7  Dsus7  Dsus7  Fsus...	1																																																																													
393 A1:   C-7  F7  Bbj7 Ebj7  D-7 Db-7  C-7 ...	1																																																																													
125 A1:   Eb F-7  G-7 F-7  G-7 F-7  F#-7 B7  F-7 ...	3																																																																													
264 A1:   C  A7  D7  D7  G7  G7  C  ...	1																																																																													
314 A1:   E7alt  E7alt  Fj7  Fj7  E7alt  ...	6																																																																													
440 A1:   G-79 F-79  G-79 F-79  C-79 Bb-79  Gb79  ...	1																																																																													
231 A1:   Ebj7  Gm7b5 C7  F-7  Db7  Ab-7 Db7...	1																																																																													
198 A1:   Fsus7913  Fsus7913  Fsus7913  Gbsu...	8																																																																													
333 A1:   D79b  D79b   \nA2:   G-7  Am7b5 D7...	3																																																																													
181 A1:   Bb7  Eb7  Ab6  Ab6  Eb-7  Ab7  ...	2																																																																													

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

[9]:	solos.head(10)																														
[9]:	<table border="1"> <thead> <tr> <th>eventid</th><th>melid</th><th>onset</th><th>pitch</th><th>duration</th><th>period</th><th>division</th><th>bar</th><th>beat</th><th>\</th></tr> </thead> <tbody> <tr><td>0</td><td>1</td><td>1</td><td>10.343492</td><td>65.0</td><td>0.138776</td><td>4</td><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>2</td><td>1</td><td>10.637642</td><td>63.0</td><td>0.171247</td><td>4</td><td>4</td><td>0</td><td>2</td></tr> </tbody> </table>	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
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																						
	(continues on next page)																														

(continued from previous page)

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
5	6	1	11.551927	58.0	0.188662	4	1	1	1
6	7	1	11.859592	58.0	0.481814	4	1	1	2
7	8	1	14.535692	50.0	0.159637	4	1	3	3
8	9	1	14.799819	57.0	0.145125	4	2	3	4
9	10	1	14.973968	60.0	0.110295	4	2	3	4
<hr/>									
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	
5	1	...		0.534657	67.636708	7.635221	0.411765	0.359536	
6	1	...	vibrato	0.584914	63.659343	5.518070	0.068182	0.403372	
7	1	...		-0.129185	58.507975	5.020340	0.133333	0.368384	
8	1	...		0.599931	71.173670	2.938194	0.285714	0.551884	
9	2	...		0.484532	69.632891	2.325457	0.600000	0.508617	
<hr/>									
0	1.056169	37.794261	f0_range	12.932532	f0_freq_hz	f0_med_dev			
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	1.049956	39.368720		6.589582		16.146429			
6	0.983151	39.429103		5.406750		11.239471			
7	0.927912	174.398513			NaN	25.203232			
8	1.064195	27.066543		7.758283		25.736430			
9	1.038483	17.141304		11.184763		15.693739			

[10 rows x 26 columns]

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

[10]:	solos.tail()
[10]:	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
	<hr/>
	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
	<hr/>
	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

(continues on next page)

(continued from previous page)

```
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.

```
[11]: solos.shape
```

```
[11]: (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.

```
[13]: solos.columns
```

```
[13]: 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.

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

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

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

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

```
[16]:   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  ...  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
```

(continues on next page)

(continued from previous page)

```
4 11.444363    8.329975    6.377737 Art Pepper
```

[5 rows x 27 columns]

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

```
[17]:   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_relpovs \
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?

```
[20]: def notelist(melid):
    solo = solos[solos["melid"] == melid]
    solo = solo[["pitch", "duration"]]
    solo["onset"] = solo["duration"].cumsum()
    return solo
```

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

```
[21]:   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
```

(continues on next page)

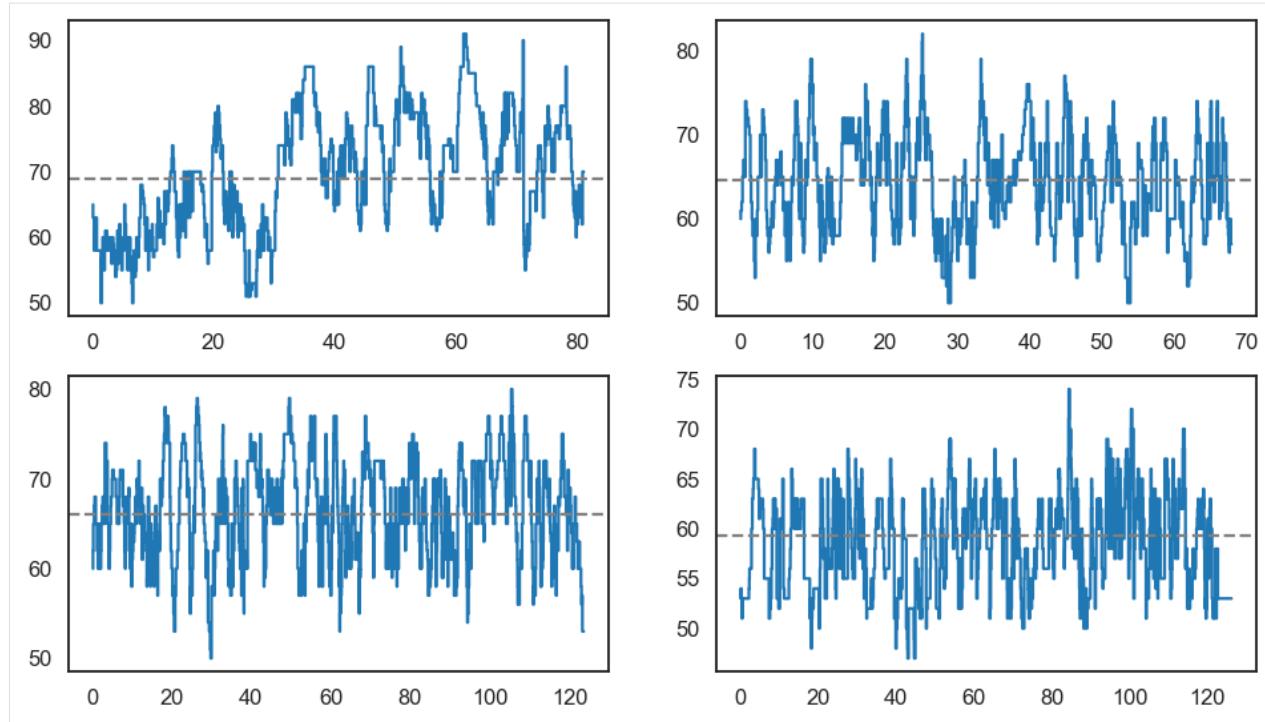
(continued from previous page)

```
529    70.0  0.187211  81.148821
```

```
[530 rows x 3 columns]
```

```
[45]: 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="--")
```

```
[46]: fig, axes = plt.subplots(2,2, figsize=(16,9))  
axes = axes.flatten()  
  
plot_melodic_profile(notelist[1], ax=axes[0], mean=True)  
plot_melodic_profile(notelist[10], ax=axes[1], mean=True)  
plot_melodic_profile(notelist[50], ax=axes[2], mean=True)  
plot_melodic_profile(notelist[100], ax=axes[3], mean=True)
```



```
[47]: 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
```

```
[48]: standardize(notelist(5))
```

	pitch	duration	onset	Rel. MIDI Pitch	Rel. Duration	Rel. Onset
2134	62.0	0.187755	0.187755	-0.485734	0.001194	0.001194
2135	63.0	0.261224	0.448980	-0.323822	0.001662	0.002856
2136	64.0	0.232200	0.681179	-0.161911	0.001477	0.004333
2137	60.0	0.092880	0.774059	-0.809556	0.000591	0.004923
2138	65.0	1.120363	1.894422	0.000000	0.007126	0.012050
...	...	...	...	...	...	...
2720	70.0	0.150000	156.117506	0.809556	0.000954	0.993003
2721	72.0	0.162245	156.279751	1.133378	0.001032	0.994035
2722	68.0	0.196939	156.476689	0.485734	0.001253	0.995288
2723	63.0	0.177551	156.654240	-0.323822	0.001129	0.996417
2724	63.0	0.563265	157.217506	-0.323822	0.003583	1.000000

[591 rows x 6 columns]

```
[53]: fig, ax = plt.subplots(figsize=(15,5))
```

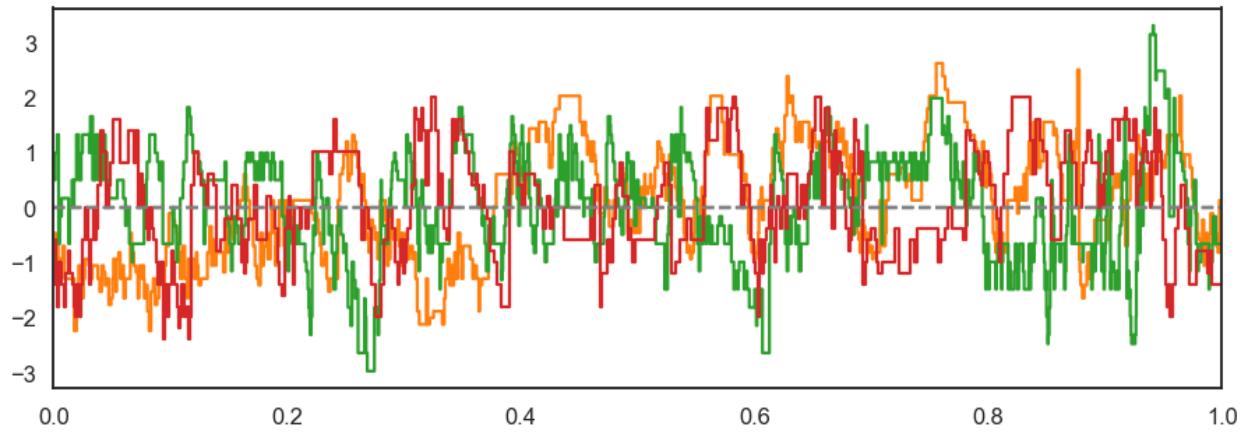
```
for i in range(4):
```

(continues on next page)

(continued from previous page)

```
plot_melodic_profile(standardize(notelist(i)),
                      mean=True,
                      standardized=True)

plt.xlim(0, 1)
plt.show()
```



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

```
[81]: big_df.head()
```

	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

```
[93]: %%time
```

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

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

(continues on next page)

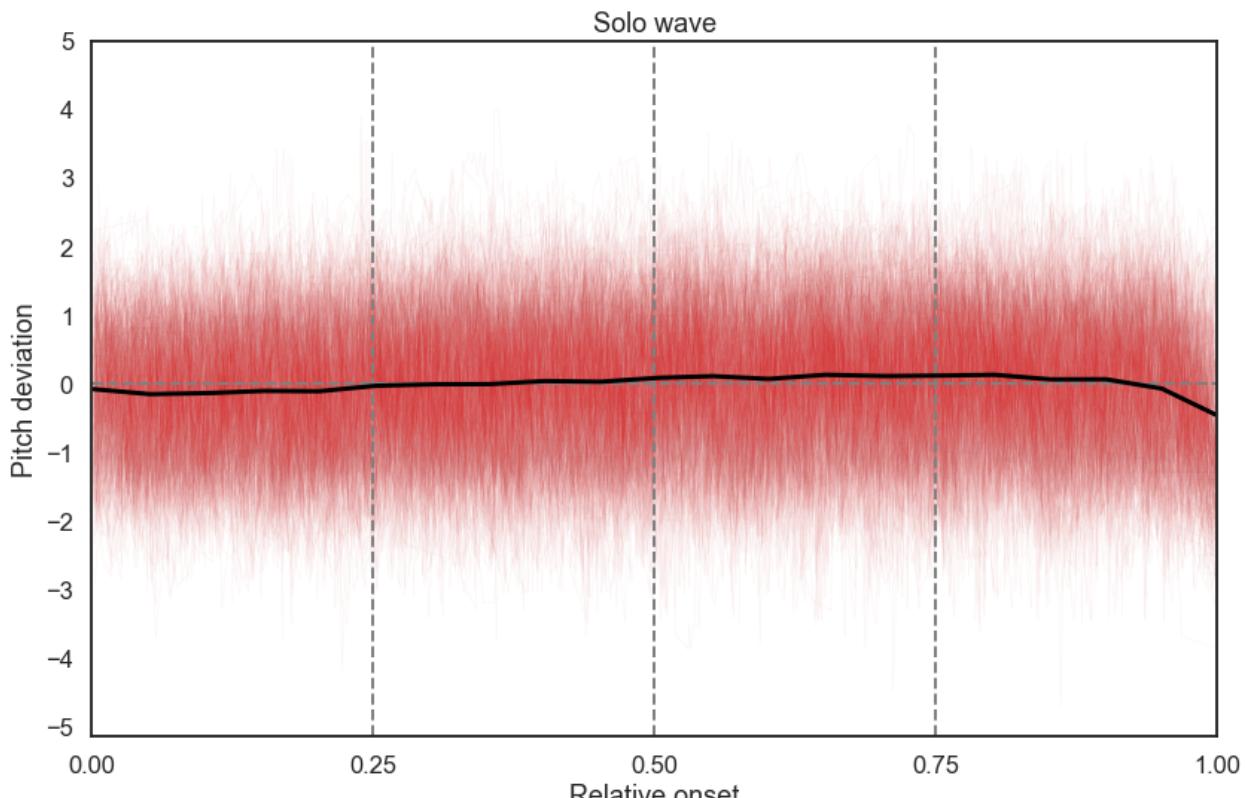
(continued from previous page)

```

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: 4.28 s

## 4.2 Pitch vs loudness

To facilitate the analyses later, we only keep the rows of the DataFrame where both the `pitch` and the `loud_med` columns have an entry (they are “not null”).

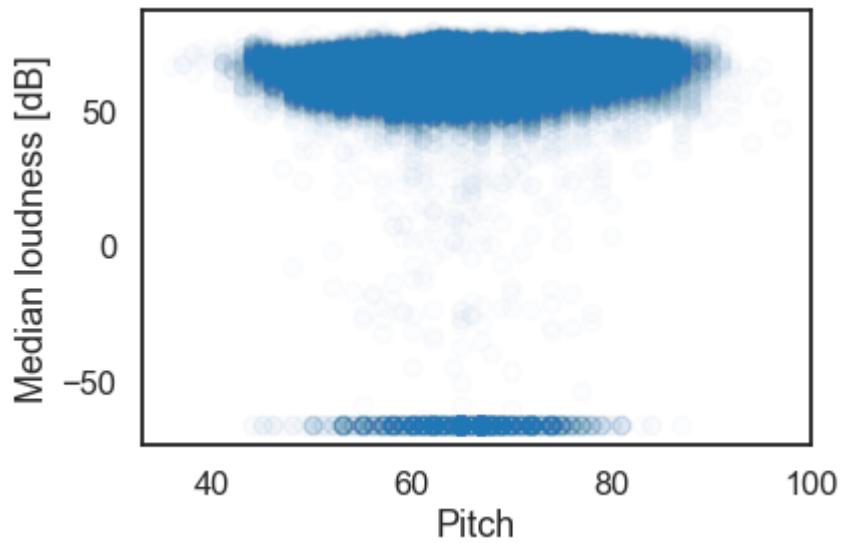
```
[15]: solos = solos[solos.pitch.notnull() & solos.loud_med.notnull()]
```

## 4.3 The “rain cloud” of Jazz solos

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

fig, ax = plt.subplots(figsize=(6,4))
ax.scatter(x,y, alpha=0.01)

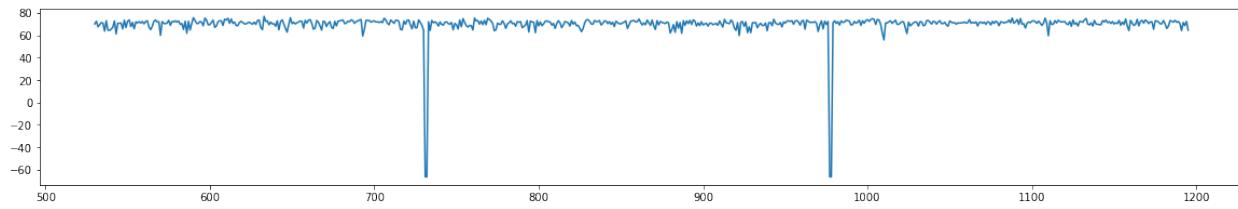
plt.xlabel("Pitch")
plt.ylabel("Median loudness [dB]")
plt.show()
```



```
[74]: solos["melid"].max()
```

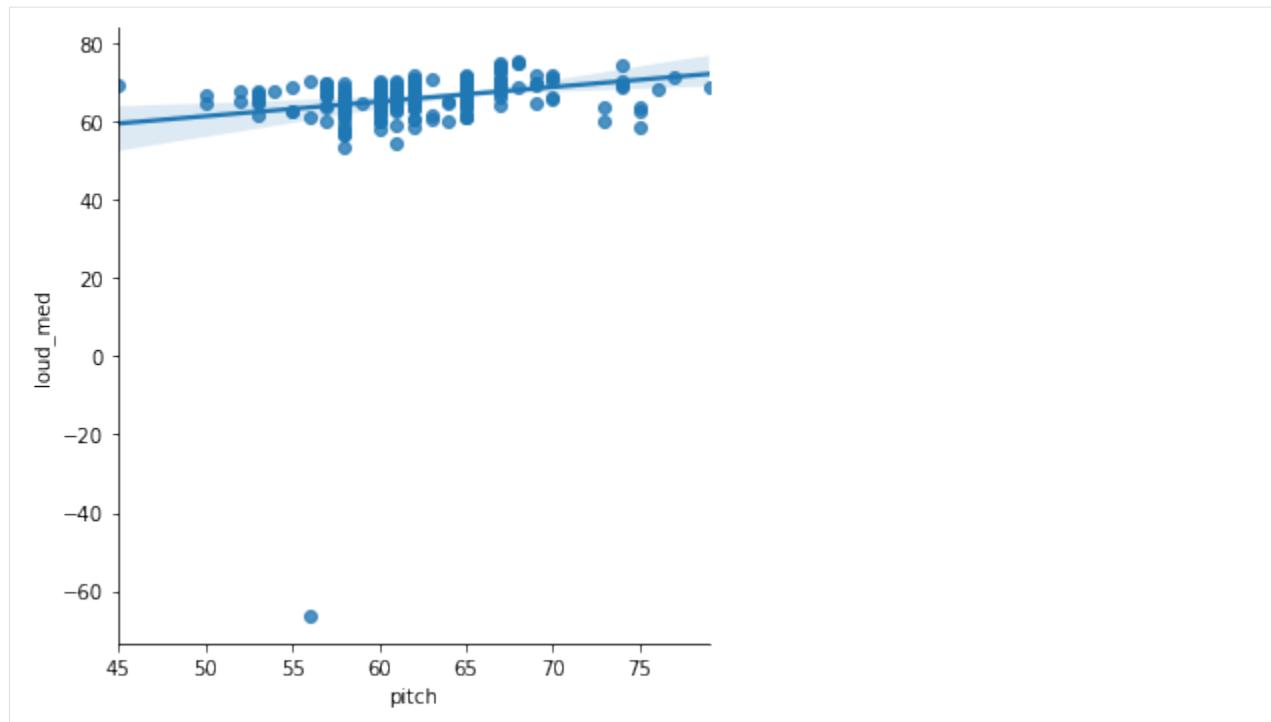
```
[74]: 456
```

```
[75]: solos[ solos["melid"] == 2 ]["loud_med"].plot(figsize=(20,3));
```



```
[135]: # fig, axes = plt.subplots(5,1, figsize=(15,20))
```

```
# for idx, rand in enumerate(np.random.randint(1,solos["melid"].unique().max(), 5)):
sns.lmplot(data=solos[ solos["melid"] == 130 ], x="pitch", y="loud_med");
```



## 4.4 Comparing performers

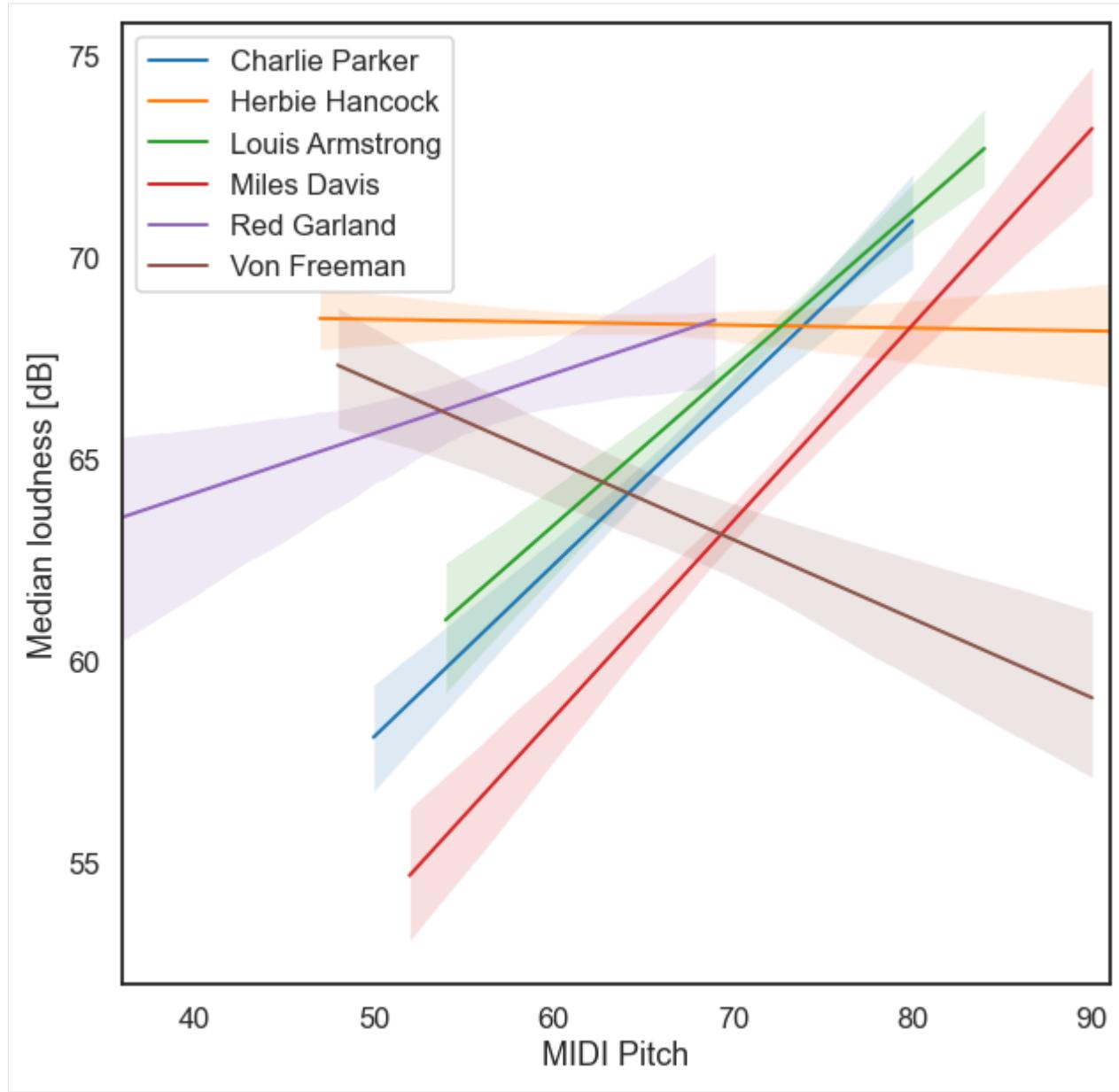
```
[16]: grouped_df = solos.groupby("performer")

[41]: selected = ["Charlie Parker", "Miles Davis", "Louis Armstrong", "Herbie Hancock",
   ↪"Von Freeman", "Red Garland"]

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

for performer, df in grouped_df:
    if performer in selected:
        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
        )

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?