In [1]:
```python
from API import DualizationDatasetAPI
```

Dataset Overview

NOTE 1: The hierarchy of the folder should be kept as is. Also, the folder/filenames should not be modified. That said, you can remove any folders (not individual files) that you do not want to be included in the dataset.

The folder containing the midi information is structured as follows:

```
       Directory: e.g.midi_files
        |
        |
        |-----Repetitions (three repetitions of the same drum pattern dualized at random times)
        |          |
        |          |--------------tested_with_four_participants
        |          |                 |-----------------------[001 P1P2P3P4] drummer* --> (13 files)
        |          |                 |                                 |------- original.mid -->
    (drum pattern used for dualization)
        |          |                 |                                 |-------
    Participant_X_repetition_Y.mid --> (dualization of the original.mid)
        |          |                ...
        |          |                 |
        |          |                 |-----------------------[001 P1P2P3P4] drummer*
        |          |
        |          |--------------tested_with_Participant_1_Only
        |                            |-----------------------[025 P1] drummer* --> (4 files)
        |                            |                                |------- original.mid --> (drum
    pattern used for dualization)
        |                            |                                |-------
    Participant_1_repetition_0.mid
        |                            |                                |-------
    Participant_1_repetition_1.mid
        |                            |                                |-------
    Participant_1_repetition_2.mid
        |                           ...
        |                            |-----------------------[072 P1] drummer1*
        |
        |
        |-----SimpleComplex (drum pattern dualized sequentially first simple then complex)
        |          |
        |          |--------------tested_with_two_participants
        |          |                 |-----------------------[073 P1P2] drummer* --> (5 files)
        |          |                 |                                |------- original.mid --> (drum
    pattern used for dualization)
        |          |                 |                                |-------
    Participant_1_simple.mid
        |          |                 |                                |-------
    Participant_1_complex.mid
        |          |                 |                                |-------
    Participant_2_simple.mid
        |          |                 |                                |-------
    Participant_2_complex.mid
        |          |                ...
        |          |                 |
        |          |                 |-----------------------[141 P1P2] drummer*
        |          |
        |          |--------------tested_with_Participant_1_Only
        |                            |-----------------------[142 P1] drummer1* --> (3 files)
        |          |                 |                                |------- original.mid --> (drum
    pattern used for dualization)
        |          |                 |                                |-------
    Participant_1_simple.mid
        |          |                 |                                |-------
    Participant_1_complex.mid
        |                           ...
        |                            |-----------------------[345 P1] drummer1*
```

NOTE 2: *drummer* refers the drummer who played the original.mid file, that is, the drummer used for obtaining the Groove Midi Dataset. In our dataset, we refer to the drummers as *Participants*

Loading the dataset

To load the dataset, you need to specify the folder where the midi files are located.

In [2]:
```
FullDualizationDataset = DualizationDatasetAPI(midi_folder="midi_files")
FullDualizationDataset.summary_dataframe.to_csv("midi_files/summary.csv", index=False)
```

```
Found 345 tested patterns
```

Many information are readily available in the dataset. To see the available information, either `print` the dataset instance or use the `summary_dict` attribute. Moreover, this information was also available as a pandas dataframe. To obtain the dataframe, use the `summary_dataframe` attribute.

Moreover, all this information is also available in a csv file located at `midi_files/summary.csv`

In [3]:
```
print(FullDualizationDataset)
```

```
------------------------------------------------------------
Number of Drum Patterns Dualized --> 345
fields available: Index(['Test Number', 'Was Tested On P1', 'Was Tested On P2',
       'Was Tested On P3', 'Was Tested On P4',
       'Was Tested On Multiple Participants', 'Test Type', 'Style', 'Tempo',
       'GMD Drummer', 'GMD Performance Session', 'GMD Segment Type',
       'GMD Segment Meter', 'Selected 2Bars From Start',
       'Dualized Midifolder Path', 'Test Folder'],
      dtype='object')
------------------------------------------------------------
```

Accessing Specific Dualizations

#

In [4]:
```
print(FullDualizationDataset.summary_dict[100])
```

```
{'Test Number': '101', 'Was Tested On P1': True, 'Was Tested On P2': True, 'Was Tested On P3': False, 'Was Tested
On P4': False, 'Was Tested On Multiple Participants': True, 'Test Type': 'Simple Complex', 'Style': 'rock', 'Tempo
': 100.0, 'GMD Drummer': 'drummer6', 'GMD Performance Session': 'session3', 'GMD Segment Type': 'beat', 'GMD Segme
nt Meter': '4-4', 'Selected 2Bars From Start': 26, 'Dualized Midifolder Path': 'midi_files/SimpleComplex/tested_wi
th_two_participants/dualized', 'Test Folder': 'midi_files/SimpleComplex/tested_with_two_participants/[101 P1P2] dr
ummer6-session3-3_rock_100_beat_4-4_best_2bar_segment_26'}
```

Access 1 or mode dualizations by using indexing. For example, to access the 100th dualization, use `FullDualizationDataset[100]` . To access the 100th and 101th dualizations, use `FullDualizationDataset[100, 101]` . To access the 100th, 101th, and 102th dualizations, use `FullDualizationDataset[100, 102, 103]` . And so on.

In [5]:
```
FullDualizationDataset[100]
```

Out[5]:
```
<API.DualizationTest at 0x7fc681593160>
```

In [6]:
```
FullDualizationDataset[100, 101]
```

Out[6]:
```
[<API.DualizationTest at 0x7fc681593220>,
 <API.DualizationTest at 0x7fc681593a60>]
```

In [7]:
```
FullDualizationDataset[list(range(100, 103))]
```

Out[7]:
```
[<API.DualizationTest at 0x7fc681601c10>,
 <API.DualizationTest at 0x7fc681601cd0>,
 <API.DualizationTest at 0x7fc681601f70>]
```

If you request a dualization that does not exist, you will get an error message indicating that the dualization does not exist and also you informed about the test numbers that are available.

In [8]:
```
# FullDualizationDataset[3000]
```

Creating Subsets of the Dataset

There are a number of methods available that allows for creating subsets of the dataset. Each of these methods returns a new instance of the `DualizationDatasetAPI` class. As such, the methods/properties can be nested to create more complex subsets as desired. Below is the list of methods/properties available:

```
Properties:

        MultipleParticipantSubset --> subset of the dataset where the dualizations were tested
with more than one participant
        SingleParticipantSubset --> subset of the dataset where the dualizations were tested with
only one participant
        ThreeRepetitionSubset --> subset of the dataset where the dualizations were dualized
three times at random times
        SimpleComplexSubset --> subset of the dataset where the dualizations were dualized
sequentially first simple then complex
        P1Subset --> subset of the dataset where the dualizations were tested with Participant 1
        P2Subset --> subset of the dataset where the dualizations were tested with Participant 2
        P3Subset --> subset of the dataset where the dualizations were tested with Participant 3
        P4Subset --> subset of the dataset where the dualizations were tested with Participant 4

Methods:
        get_subset_matching_styles(styles, hard_match) --> subset of the dataset where the
dualizations were dualized with the specified styles
        get_subset_within_tempo_range(min_tempo, max_tempo) --> subset of the dataset where the
dualizations were dualized within the specified tempo range
```

In [9]:
```
# get subset of dualizations tested with more than one participant (i.e. A1 and B1 Sessions)
FullDualizationDataset.MultipleParticipantSubset.summary_dataframe
```

Out[9]:

	Test Number	Was Tested On P1	Was Tested On P2	Was Tested On P3	Was Tested On P4	Was Tested On Multiple Participants	Test Type	Style	Tempo	GMD Drummer	GMD Performance Session	GMD Segment Type	GMD Segment Meter	Selected 2Bars From Start	
0	001	True	True	True	True	True	Three Random Repetitions	jazz-swing	110.0	drummer10	session1	beat	4-4	15	/te
1	002	True	True	True	True	True	Three Random Repetitions	soul-groove4	80.0	drummer1	eval	beat	4-4	6	/te
2	003	True	True	True	True	True	Three Random Repetitions	reggae	78.0	drummer1	session1	beat	4-4	8	/te
3	004	True	True	True	True	True	Three Random Repetitions	rock-halftime	140.0	drummer1	session1	beat	4-4	10	/te
4	005	True	True	True	True	True	Three Random Repetitions	latin-samba	116.0	drummer1	session1	beat	4-4	25	/te
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
88	137	True	True	False	False	True	Simple Complex	rock	120.0	drummer9	session1	beat	4-4	7	mi /t
89	138	True	True	False	False	True	Simple Complex	rock	90.0	drummer9	session1	beat	4-4	3	mi /t
90	139	True	True	False	False	True	Simple Complex	rock	90.0	drummer9	session1	beat	4-4	0	mi /t
91	140	True	True	False	False	True	Simple Complex	rock	100.0	drummer9	session1	beat	4-4	1	mi /t
92	141	True	True	False	False	True	Simple Complex	rock	100.0	drummer9	session1	beat	4-4	0	mi /t

93 rows × 16 columns

In [10]:
```
# filter the subset of dualizations that match one or more styles
JazzDualizedDataset = FullDualizationDataset.get_subset_matching_styles(style="jazz", hard_match=False)
JazzDualizedDataset.summary_dataframe["Style"]
```

Out[10]:
```
0         jazz-swing
1          jazz-funk
2               jazz
3         jazz-swing
4         jazz-swing
5               jazz
6         jazz-march
7       jazz-klezmer
8        jazz-fusion
9        jazz-fusion
10              jazz
11    jazz-mediumfast
12    jazz-mediumfast
13        jazz-swing
14       jazz-linear
15              jazz
16        jazz-swing
17        jazz-swing
18        jazz-swing
19        jazz-swing
20        jazz-swing
21        jazz-swing
22              jazz
23              jazz
24              jazz
25         jazz-fast
26        jazz-march
27        jazz-march
28       jazz-linear
29              jazz
30              jazz
31              jazz
32       jazz-fusion
33       jazz-fusion
Name: Style, dtype: object
```

In [11]:
```
# filter the subset of dualizations that match one or more styles
JazzRockDualizedDataset = FullDualizationDataset.get_subset_matching_styles(style=["rock", "jazz"], hard_match=Fals
JazzRockDualizedDataset.summary_dataframe["Style"]
```

Out[11]:
```
0        jazz-swing
1     rock-halftime
2         jazz-funk
3              jazz
4              rock
          ...
154            rock
155            rock
156            rock
157            rock
158            rock
Name: Style, Length: 159, dtype: object
```

In [12]:
```
# or create a subset of dualizations that excludes one or more styles
NotJazzRockDualizedDataset = FullDualizationDataset.get_subset_excluding_styles(style=["rock", "jazz"], hard_match=
NotJazzRockDualizedDataset.summary_dataframe["Style"]
```

Out[12]:
```
0      soul-groove4
1            reggae
2       latin-samba
3              punk
4       dance-disco
          ...
181       highlife
182       afrobeat
183           funk
184           funk
185         hiphop
Name: Style, Length: 186, dtype: object
```

In [13]:
```
# also you can filter using the tempo range
WithinTempoDataset = JazzDualizedDataset.get_subset_within_tempo_range(min_=100, max_=120)
WithinTempoDataset.summary_dataframe["Tempo"]
```

Out[13]:
```
0    110.0
1    116.0
2    112.0
3    110.0
4    120.0
5    102.0
6    105.0
7    120.0
8    120.0
9    120.0
Name: Tempo, dtype: float64
```

Iterating Through the Dualizations and Accessing Information

Multiple ways to do this

In [14]:
```
# 1. Get the test numbers in the dataset
print("Number of tests in the dataset: ", len(FullDualizationDataset))
test_numbers = FullDualizationDataset.get_test_numbers()
for tn in test_numbers:
    dualizationTest = FullDualizationDataset[tn]
```

```
Number of tests in the dataset:  345
```

In [15]:
```
print(FullDualizationDataset.get_folder_path_for_test_number(tn),
        FullDualizationDataset.get_tested_drum_pattern_path(tn),
        FullDualizationDataset.get_test_type_for_test_number(tn),
        FullDualizationDataset.get_style_for_test_number(tn),
        FullDualizationDataset.get_tempo_for_test_number(tn))


FullDualizationDataset.get_participant_dualizations_for_test_number(tn, participant=1)
```

```
midi_files/SimpleComplex/tested_with_Participant_1_Only/[345 P1] drummer9-session1-8_rock_100_beat_4-4_best_2bar_s
egment_2 midi_files/SimpleComplex/tested_with_Participant_1_Only/[345 P1] drummer9-session1-8_rock_100_beat_4-4_be
st_2bar_segment_2/original.mid Simple Complex rock 100.0
```

Out[15]:
```
Participant 1, Dualization Test: Simple Complex, Style: rock, Tempo: 100.0
 use .simple and .complex to access dualizations,
use .original to access the drum pattern used for dualization
```

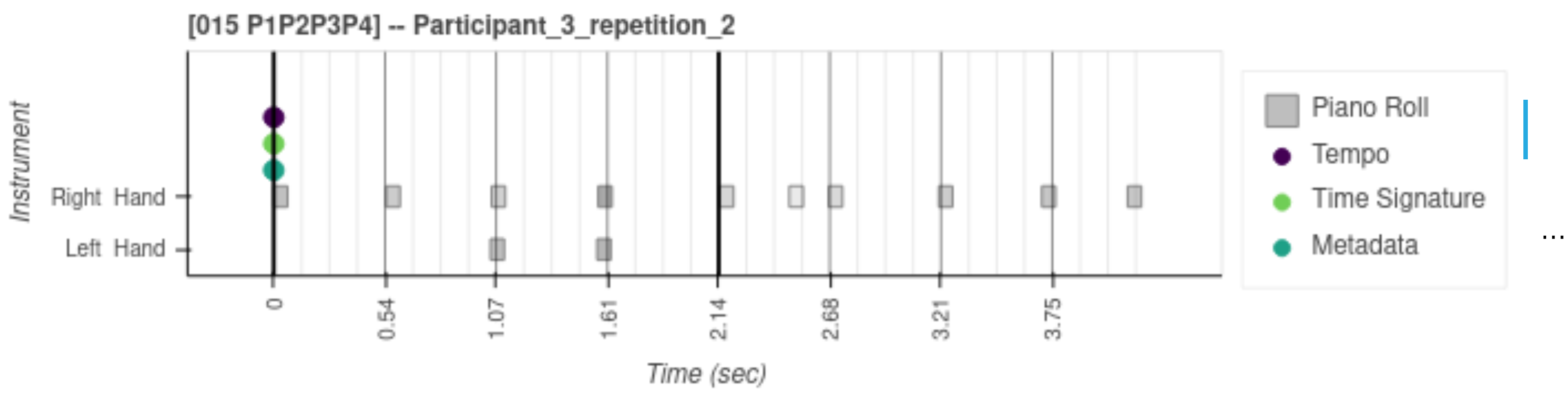
In [16]:
```
# 2. Get the tests directly from the dataset
dualization_tests = FullDualizationDataset.dualization_tests

for dualizationTest in dualization_tests:
    pass
```

Visualizing the Dualizations

Let's Plot the Last Repetition of the Third Particpant for the 15th Drum Pattern Tested

In [17]:
```
FullDualizationDataset[15].P3.rep2.piano_roll(show=True,width=800, height=200)
```

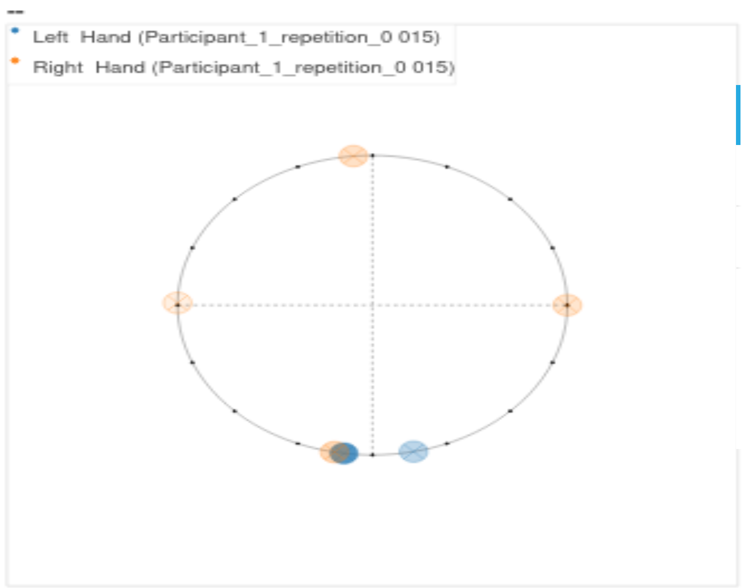


Out[17]: **Figure**(id = '1002', …)

You can also visualize the patterns in a cicular fashion

In [18]:
```
FullDualizationDataset[15].P1.rep0.z_plot(color_with_vel_values=True, quantize_all=False, scale_plot=0.8,
                                          flatten_all=False, show_figure=True)
```

Bar 1 Bar 2

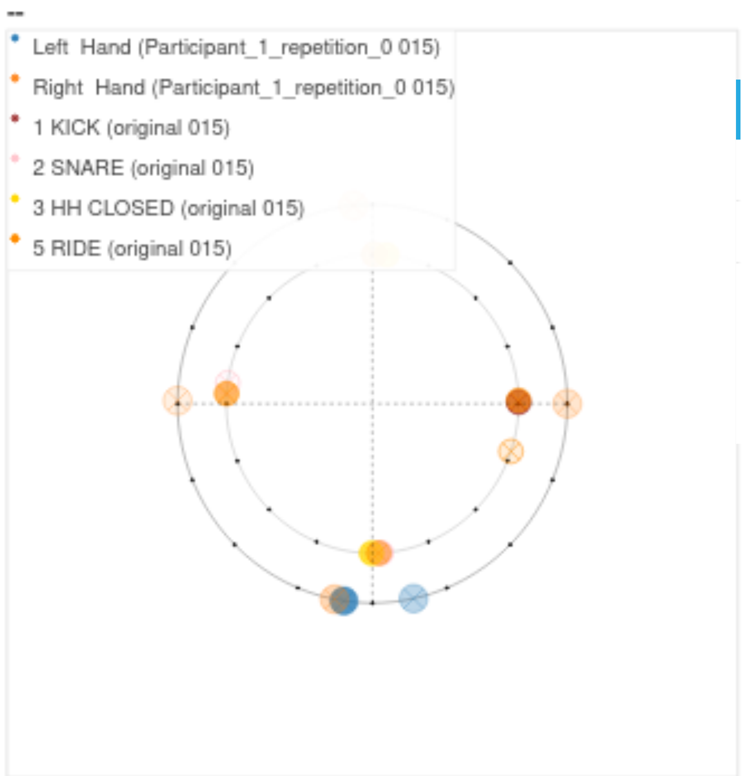


Out[18]: **Tabs**(id = '1681', …)

You can compare two patterns using the `compare_with_other_Pattern` argument Lets compare the last repetition first with the original drum pattern and then perhaps another participant dualization of the same drum pattern

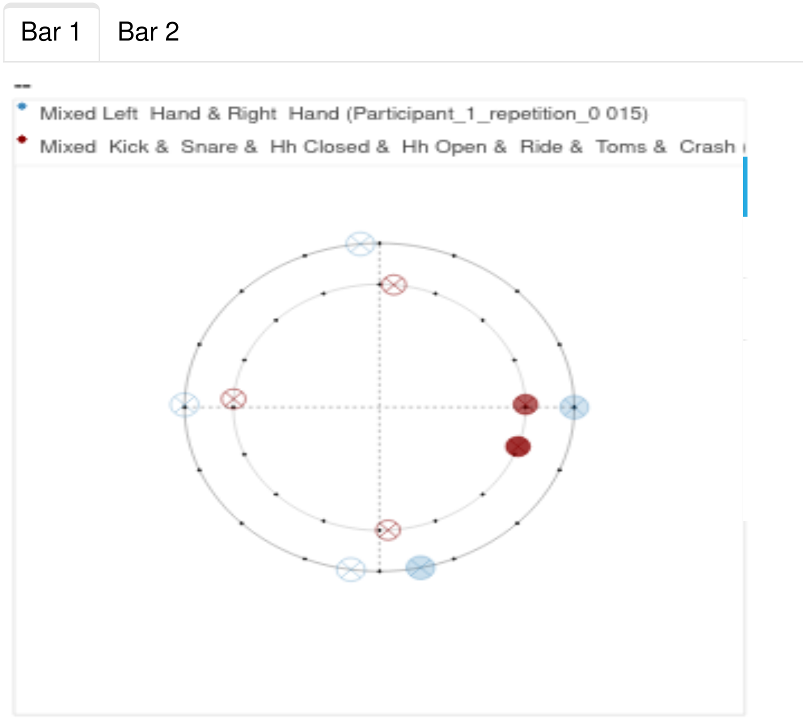
In [19]:
```
FullDualizationDataset[15].P1.rep0.z_plot(color_with_vel_values=True, quantize_all=False, scale_plot=0.8,
                                          flatten_all=False, show_figure=True, compare_with_other_Pattern=FullDuali
```

Bar 1 Bar 2



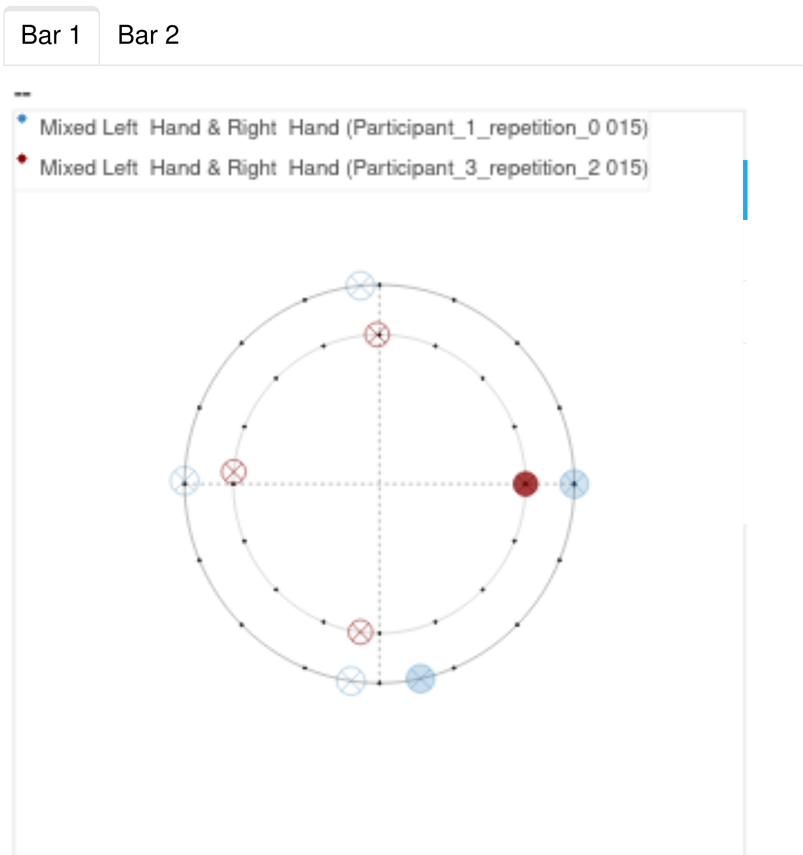
Out[19]: **Tabs**(id = '4739', …)

In [20]:
```
FullDualizationDataset[15].P1.rep0.z_plot(color_with_vel_values=True, quantize_all=False, scale_plot=0.8,
                                          flatten_all=True, show_figure=True, compare_with_other_Pattern=FullDualiz
```



Out[20]: **Tabs**(id = '7133', …)

In [21]:
```
FullDualizationDataset[15].P1.rep0.z_plot(color_with_vel_values=True, quantize_all=False, scale_plot=0.8,
                                          flatten_all=True, show_figure=True, compare_with_other_Pattern=FullDualiz
```



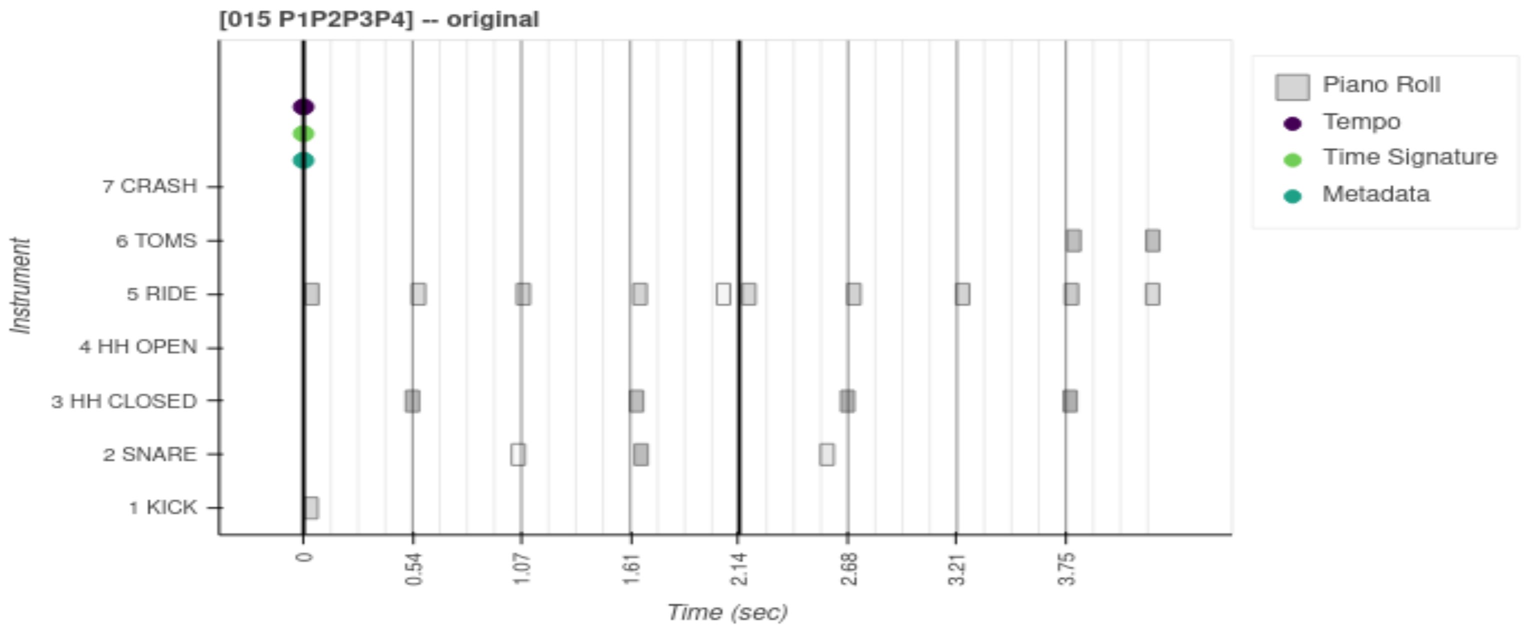
Out[21]: **Tabs**(id = '9341', …)

If you want to see all the repetitions/original pattern for a specific test and all participant use any of the two following methods to get them all in a multi Tab plot: `piano_rolls` or `z_plots` (you can use it per participant or for all participants).

In [22]:
```
FullDualizationDataset[15].piano_rolls(True)
```

Participant 1 Participant 2 Participant 3 Participant 4

[015 P1P2P3P4] Participant_1_repetition_0.mid Participant_1_repetition_1.mid Participant_1_repetition_2.mid



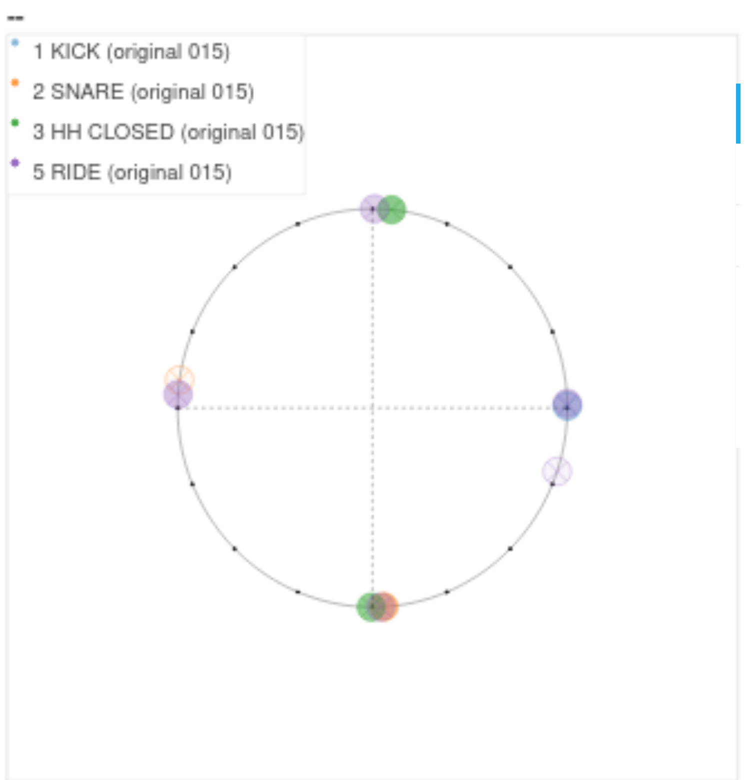
Out[22]: **Tabs**(id = '12877', …)

In [23]:
```
FullDualizationDataset[15].z_plots(show_=True, scale_plot=0.8, color_with_vel_values=True)
```

Participant 1 Participant 2 Participant 3 Participant 4

[015 P1P2P3P4] P1_rep_0 P1_rep_1 P1_rep_2

Bar 1 Bar 2



Out[23]: **Tabs**(id = '23080', …)

Accessing Midi Files

You can access the midi files in multiple ways: 1. as path in midi_files directory, 2. as a pretty_midi instance and 3. as a note_sequence instance. The following code snippets show how to access the midi files in each of these ways.

In [24]:
```
#1. as path
print(FullDualizationDataset[15].P1.rep0.path, "\n"*2,FullDualizationDataset[15].P1.original.path)
```

```
midi_files/Repetitions/tested_with_four_participants/[015 P1P2P3P4] drummer7-session1-15_jazz_112_beat_4-4_best_2bar_segment_25/Participant_1_repetition_0.mid

 midi_files/Repetitions/tested_with_four_participants/[015 P1P2P3P4] drummer7-session1-15_jazz_112_beat_4-4_best_2bar_segment_25/original.mid
```

In [25]:
```
#2. as a pretty_midi instance
pm_ = FullDualizationDataset[15].P1.rep0.pretty_midi
pm_
```

Out[25]:
```
<pretty_midi.pretty_midi.PrettyMIDI at 0x7fc68022f0a0>
```

In [26]:
```
#2. as a pretty_midi instance
ns_ = FullDualizationDataset[15].P1.rep0.note_sequence
ns_
```

Out[26]:


```
ticks_per_quarter: 96
time_signatures {
  numerator: 4
  denominator: 4
}
time_signatures {
  numerator: 4
  denominator: 4
}
tempos {
  qpm: 112.00005973336519
}
notes {
  pitch: 74
  velocity: 25
  end_time: 0.078124958333333341
}
notes {
  pitch: 74
  velocity: 30
  start_time: 0.569196125
  end_time: 0.66964250000000014
}
notes {
  pitch: 74
  velocity: 15
  start_time: 1.0658476458333335
  end_time: 1.1607136666666669
}
notes {
  pitch: 60
  velocity: 99
  start_time: 1.5569188125000002
  end_time: 1.6238830625000003
}
notes {
  pitch: 74
  velocity: 39
  start_time: 1.5401777500000002
  end_time: 1.6406241250000002
}
notes {
  pitch: 60
  velocity: 63
  start_time: 1.6238830625000003
  end_time: 1.6796866041666669
}
notes {
  pitch: 60
  velocity: 38
  start_time: 1.6796866041666669
  end_time: 1.746650854166667
}
notes {
  pitch: 60
  velocity: 14
  start_time: 1.7522312083333336
  end_time: 1.847097229166667
}
notes {
  pitch: 74
  velocity: 30
  start_time: 2.2321416666666671
  end_time: 2.3325880416666669
}
notes {
  pitch: 60
  velocity: 80
  start_time: 2.5613825625000004
  end_time: 2.6618289375000006
}
notes {
  pitch: 74
  velocity: 12
  start_time: 2.7455342500000004
  end_time: 2.8459806250000006
}
notes {
  pitch: 74
  velocity: 38
  start_time: 3.2979893125000004
  end_time: 3.3984356875000006
}
notes {
  pitch: 60
  velocity: 86
  start_time: 3.7946408333333337
  end_time: 3.8950872083333339
}
notes {
  pitch: 60
```

```
    velocity: 83
    start_time: 4.1238817291666674
    end_time: 4.21874775
  }
  total_time: 4.21874775
  control_changes {
    control_number: 16
    control_value: 80
  }
  control_changes {
    control_number: 16
    control_value: 96
  }
  control_changes {
    control_number: 17
  }
  control_changes {
    control_number: 17
    control_value: 80
  }
  control_changes {
    time: 0.53013364583333344
    control_number: 17
    control_value: 80
  }
  control_changes {
    time: 0.53013364583333344
    control_number: 17
    control_value: 84
  }
  control_changes {
    time: 0.98772268750000014
    control_number: 17
    control_value: 72
  }
  control_changes {
    time: 0.98772268750000014
    control_number: 17
    control_value: 84
  }
  control_changes {
    time: 1.4397313750000003
    control_number: 17
    control_value: 72
  }
  control_changes {
    time: 1.4397313750000003
    control_number: 17
    control_value: 80
  }
  control_changes {
    time: 1.4508920833333336
    control_number: 16
    control_value: 96
  }
  control_changes {
    time: 1.4508920833333336
    control_number: 16
    control_value: 100
  }
  control_changes {
    time: 1.5178563333333335
    control_number: 16
    control_value: 96
  }
  control_changes {
    time: 1.5178563333333335
    control_number: 16
    control_value: 100
  }
  control_changes {
    time: 1.5680795208333336
    control_number: 16
    control_value: 80
  }
  control_changes {
    time: 1.5680795208333336
    control_number: 16
    control_value: 96
  }
  control_changes {
    time: 1.6294634166666668
    control_number: 16
    control_value: 76
  }
  control_changes {
    time: 1.6294634166666668
    control_number: 16
    control_value: 80
  }
  control_changes {
```

```
  time: 2.0814721041666671
  control_number: 17
  control_value: 80
}
control_changes {
  time: 2.0814721041666671
  control_number: 17
  control_value: 84
}
control_changes {
  time: 2.3883915833333336
  control_number: 16
  control_value: 76
}
control_changes {
  time: 2.3883915833333336
  control_number: 16
  control_value: 92
}
control_changes {
  time: 2.5613825625000004
  control_number: 17
  control_value: 76
}
control_changes {
  time: 2.5613825625000004
  control_number: 17
  control_value: 84
}
control_changes {
  time: 3.0803555000000005
  control_number: 17
  control_value: 76
}
control_changes {
  time: 3.0803555000000005
  control_number: 17
  control_value: 80
}
control_changes {
  time: 3.5435248958333339
  control_number: 16
  control_value: 92
}
control_changes {
  time: 3.5435248958333339
  control_number: 16
  control_value: 96
}
control_changes {
  time: 3.8504443750000004
  control_number: 16
  control_value: 88
}
control_changes {
  time: 3.8504443750000004
  control_number: 16
  control_value: 96
}
source_info {
  encoding_type: MIDI
  parser: PRETTY_MIDI
}
instrument_infos {
  name: "MIDI 7\000"
}
```


Accessing the Scores as arrays

You can directly access the scores as arrays with certain limitations. At this point, only relative to a 16th note quantization in 4-4 meter. The following code snippets show how to access the scores as arrays.

the arrays in this case will be shaped as (T, N) where T is the number of grid lines and N refers to number of voices at each step

In [27]:
```
hits = FullDualizationDataset[15].P1.rep0.hvo_sequence.hits
print(hits.shape, hits.transpose())
```

```
(32, 2) [[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.
  0. 0. 0. 0. 1. 0. 0. 1.]
 [1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0.
  0. 1. 0. 0. 0. 0. 0. 0.]]
```

In [28]:
```
velocities = FullDualizationDataset[15].P1.rep0.hvo_sequence.velocities
print(velocities.shape, velocities.transpose())
```

```
(32, 2) [[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.77952756 0.2992126  0.         0.         0.         0.
  0.         0.62992126 0.         0.         0.         0.
  0.         0.         0.         0.         0.67716535 0.
  0.         0.65354331]
 [0.19685039 0.         0.         0.         0.23622047 0.
  0.         0.         0.11811024 0.         0.         0.
  0.30708661 0.         0.         0.         0.         0.23622047
  0.         0.         0.         0.09448819 0.         0.
  0.         0.2992126  0.         0.         0.         0.
  0.         0.        ]]
```

In [29]:
```
offsets = FullDualizationDataset[15].P1.rep0.hvo_sequence.offsets   # offsets from -0,5 to 0,5 where +/-0.5 means 3
print(offsets.shape, offsets.transpose())
```

```
(32, 2) [[ 0.     0.     0.     0.     0.     0.     0.     0.     0.     0.
   0.     0.    -0.374 -0.458  0.     0.     0.     0.     0.     0.123
   0.     0.     0.     0.     0.     0.     0.     0.     0.333  0.
   0.    -0.21 ]
 [ 0.     0.     0.     0.     0.248  0.     0.     0.    -0.039  0.
   0.     0.    -0.499  0.     0.     0.     0.    -0.335  0.     0.
   0.    -0.496  0.     0.     0.    -0.373  0.     0.     0.     0.
   0.     0.   ]]
```

If you need a single voice array corresponding to the flattened score, you can use the following method

In [30]:
```
hits = FullDualizationDataset[15].P1.rep0.hvo_sequence.flatten_voices(reduce_dim=True)[:, 0]
velocities = FullDualizationDataset[15].P1.rep0.hvo_sequence.flatten_voices(reduce_dim=True)[:, 1]
offsets = FullDualizationDataset[15].P1.rep0.hvo_sequence.flatten_voices(reduce_dim=True)[:, 2]

print(hits.shape, hits)
print(velocities.shape, velocities)
print(offsets.shape, offsets)
```

```
(32,) [1. 0. 0. 0. 1. 0. 0. 0. 1. 0. 0. 0. 1. 1. 0. 0. 0. 1. 0. 1. 0. 1. 0. 0.
 0. 1. 0. 0. 1. 0. 0. 1.]
(32,) [0.19685039 0.         0.         0.         0.23622047 0.
 0.         0.         0.11811024 0.         0.         0.
 0.77952756 0.2992126  0.         0.         0.         0.23622047
 0.         0.62992126 0.         0.09448819 0.         0.
 0.         0.2992126  0.         0.         0.67716535 0.
 0.         0.65354331]
(32,) [ 0.     0.     0.     0.     0.248  0.     0.     0.    -0.039  0.
  0.     0.    -0.374 -0.458  0.     0.     0.    -0.335  0.     0.123
  0.    -0.496  0.     0.     0.    -0.373  0.     0.     0.333  0.
  0.    -0.21 ]
```

Calculating Metrics

A number of measures are available for calculating the similarity between the original and dualized patterns. These measures are calculated for each of the repetitions and the original pattern. The following code snippets show how to access these measures.

In [31]:
```
print(FullDualizationDataset[4].P1.calculate_intra_dualization_edit_distances())
print(FullDualizationDataset[4].P1.calculate_intra_dualization_jaccard_similarities())
print(FullDualizationDataset[4].P1.calculate_intra_dualization_jaccard_similarities())
print(FullDualizationDataset[4].P1.calculate_dualization_to_original_edit_distances())
print(FullDualizationDataset[4].P1.calculate_dualization_to_original_jaccard_similarities())
```

```
[8, 6, 4]
[0.31, 0.36, 0.71]
[0.31, 0.36, 0.71]
[8, 2, 3]
[0.29, 0.8, 0.79]
```

In [32]:
```
print(FullDualizationDataset[4].P1.calculate_inter_dualization_jaccard_similarities(FullDualizationDataset[4].P4))
print(FullDualizationDataset[4].P1.calculate_inter_dualization_edit_distances(FullDualizationDataset[4].P4))
```

```
[0.29, 0.23, 0.27, 0.64, 0.5, 0.6, 0.62, 0.73, 0.69]
[7, 7, 7, 5, 7, 5, 5, 3, 4]
```

In [ ]: