Worksheet 7

You are encouraged to work in groups of up to 3 total students, but each student should make their own submission on Canvas. (It's fine for everyone in the group to have the same upload.)

In [1]:
```
import pandas as pd
import altair as alt
import numpy as np
```

Preparing the data

- Import the attached Spotify dataset as `df` using the pandas function `read_csv`.

In [2]:
```
df = pd.read_csv("../Data/spotify_dataset.csv")
df.head()
```

Out[2]:

	Index	Highest Charting Position	Number of Times Charted	Week of Highest Charting	Song Name	Streams	Artist	Artist Followers	
0	1	1	8	2021-07-23--2021-07-30	Beggin'	48,633,449	Måneskin	3377762	3W
1	2	2	3	2021-07-23--2021-07-30	STAY (with Justin Bieber)	47,248,719	The Kid LAROI	2230022	5HCyW
2	3	1	11	2021-06-25--2021-07-02	good 4 u	40,162,559	Olivia Rodrigo	6266514	4ZtFar
3	4	3	5	2021-07-02--2021-07-09	Bad Habits	37,799,456	Ed Sheeran	83293380	6PQ88
4	5	5	1	2021-07-23--2021-07-30	INDUSTRY BABY (feat. Jack Harlow)	33,948,454	Lil Nas X	5473565	27Nov

5 rows × 23 columns

- Look at the first few rows in the DataFrame. Which columns look like they *should* be numeric?

The columns that look like they should be numeric are Highest Charting Position, Number of Times Charted, Streams, Artist Followers, Popularity, and the remainder Danceability though Valence.

- Evaluate `df.dtypes`, and notice that only a few of these columns are actually numeric.

In [3]:
```
df.dtypes
```

Out[3]:
```
Index                          int64
Highest Charting Position      int64
Number of Times Charted        int64
Week of Highest Charting      object
Song Name                     object
Streams                       object
Artist                        object
Artist Followers              object
Song ID                       object
Genre                         object
Release Date                  object
Weeks Charted                 object
Popularity                    object
Danceability                  object
Energy                        object
Loudness                      object
Speechiness                   object
Acousticness                  object
Liveness                      object
Tempo                         object
Duration (ms)                 object
Valence                       object
Chord                         object
dtype: object
```

The problem is that missing values are specified in an unusual way in this dataset, and pandas does not recognize them as missing.

- Evaluate the function `pd.to_numeric` on the "Energy" column in `df`. Look at the error message. It is telling you what the problematic values are.

In [4]:
```
#pd.to_numeric(df["Energy"])
```

(There is a better way to do this, using `df.replace`, but the approach suggested here gives some practice with `applymap`, with lambda functions, and with the Python operator `??? if ??? else ???`.)

- Using the following template, replace these problematic values with the Not a Number value defined in NumPy. If the input `x` is not missing, it should be left unchanged.

  ```
  df2 = df.applymap(lambda x: ??? if x == ??? else ???)
  ```

- Define `first_col = "Popularity"` and `last_col = "Valence"`.

In [5]:
```
df2 = df.map(lambda x: np.nan if x == " " else x)
```

In [6]:
```
first_col = "Popularity"
last_col = "Valence"
```

- Using `apply` and a suitable `axis` argument, apply the `pd.to_numeric` function to each column from `first_col` to `last_col`.

(Comment 0. Use `df2.loc[???] = df2.loc[???].apply(???, axis=???)` so that the values in `df2` get changed. Comment 1. Slicing with `loc` is one of the few examples in Python where the right endpoint *is* included. Comment 2. Neither the column labels nor the row labels are getting changed in this case, so my explanation for whether to use `axis=0` or `axis=1` does not quite apply. But imagine we were applying `sum` instead of `pd.to_numeric`, that would collapse the whole column down to a single number, and so the row labels would be changing.)

In [7]:
```
cols = df2.loc[:, first_col:last_col].columns
for col in cols:
    df2[col] = pd.to_numeric(df2[col])
```

- Import the `is_numeric_dtype` and `is_string_dtype` functions from `pandas.api.types`. (You can do these both on the same line, separating the function names with commas.)

In [8]:
```
from pandas.api.types import is_numeric_dtype, is_string_dtype
```

- Using list comprehension and the function `is_numeric_dtype`, make a list of all the column names in `df2` which have a numeric dtype now. Name this list `num_list`. (This list `num_list` should be a list of strings, not a list of pandas Series.)

In [9]:
```
num_list = [col for col in df2.columns if is_numeric_dtype(df2[col])]
num_list
```

Out[9]:
```
['Index',
 'Highest Charting Position',
 'Number of Times Charted',
 'Popularity',
 'Danceability',
 'Energy',
 'Loudness',
 'Speechiness',
 'Acousticness',
 'Liveness',
 'Tempo',
 'Duration (ms)',
 'Valence']
```

- Check your answer: `num_list` should have length `13` and the `type` of `num_list[3]` should be `str`.

In [10]:
```
print(len(num_list))
print(type(num_list[3]))
```

```
13
<class 'str'>
```

- Drop the rows which contain missing values, using the code `df3 = df2.dropna(axis=???)`.

In [11]:
```
df3 = df2.dropna(axis=0)
```

- Check your answer. The DataFrame `df3` should have `1545` rows and `23` columns.

In [12]:
```
df3.shape
```

Out[12]: (1545, 23)

Plotting the data

- Write a function `make_chart` which takes as input the name of a column `c` in `df3`, and as output returns an Altair scatter plot chart using `"Energy"` for the x-axis, using column `c` for the y-axis, and using `"Danceability"` for color.

In [13]:
```
def make_chart(c):
    return alt.Chart(df3).mark_circle().encode(x = "Energy", y = c, color = "Dancea
```

- Adjust the color scale to something more colorful in your definition of `make_chart`. You can either use the old syntax `scale=alt.Scale(scheme=???)` or the new syntax `.scale(scheme=???)`, but if you want to use the new syntax, make sure you pip install the most recent version of Altair before importing Altair (you can always restart the notebook if you need to). Options for color scheme

In [14]:
```
def make_chart(c):
    return alt.Chart(df3).mark_circle().encode(x = "Energy", y = c, color = alt.Col
```

- Using list comprehension and the list `num_list` you made above, make a new list `chart_list` which contains `make_chart(c)` for every `c` in `num_list`.

In [15]:
```
chart_list = [chart for chart in [make_chart(c) for c in num_list]]
```

- Use the following code to make a new Altair chart, which has all of the previous charts vertically concatenated together.

(Comment. This `*` before `chart_list` is performing what is called "list unpacking". Rather than giving `alt.vconcat` a single input value, a list, we are giving it 13 separate input values. In other words, `alt.vconcat(*chart_list)` is the same as `alt.vconcat(chart_list[0], chart_list[1], ..., chart_list[12])`.)

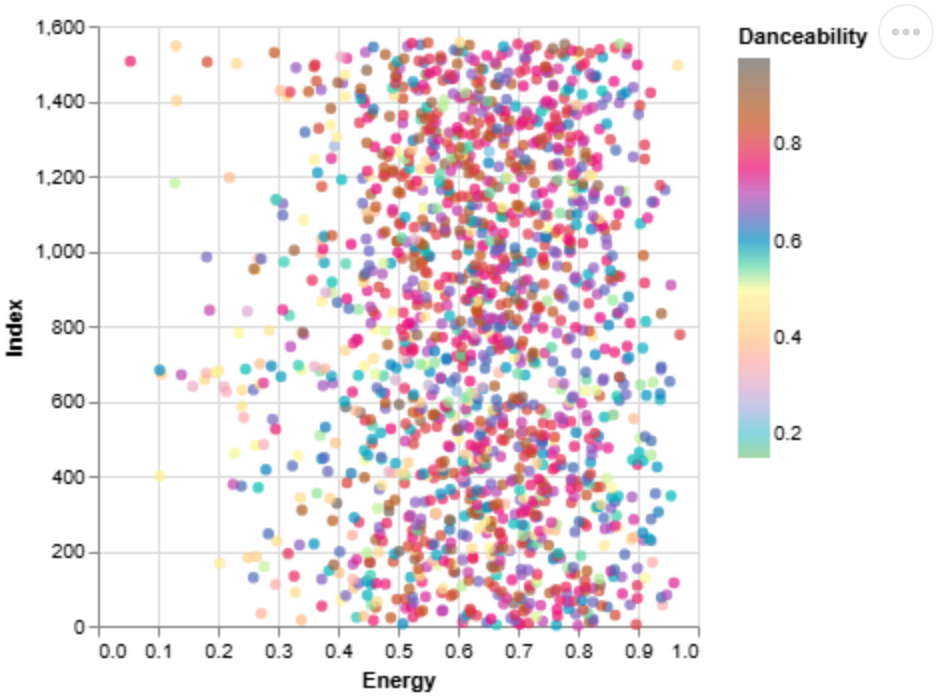
```
total_chart = alt.vconcat(*chart_list)
```

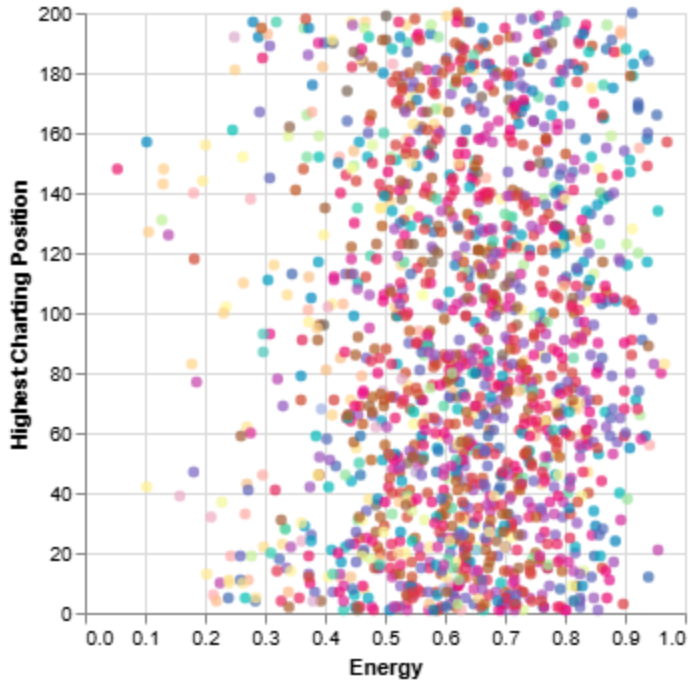
In [16]:
```
total_chart = alt.vconcat(*chart_list)
```

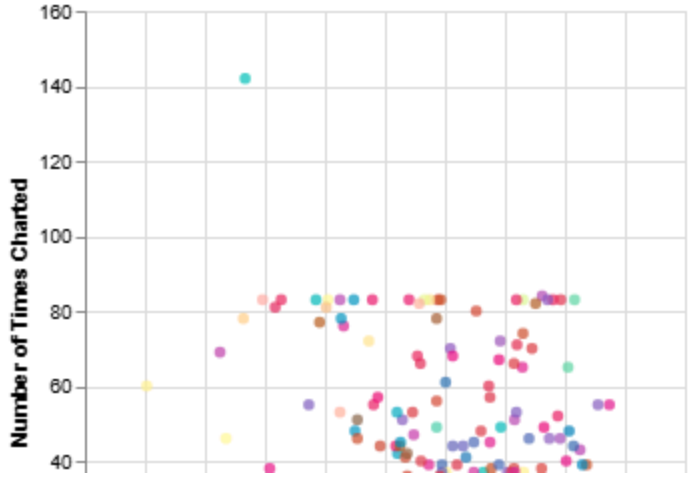
- Display this `vconcat` chart (just evaluate `total_chart` in its own cell) and make sure it looks reasonable. You should see 13 distinct Altair charts, arranged vertically.

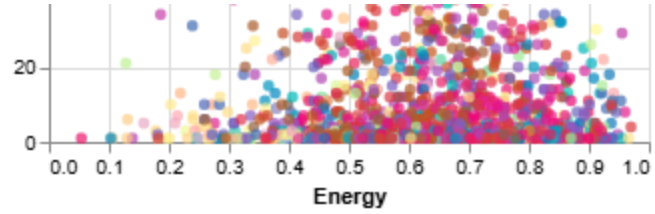
In [17]:
```
total_chart
```

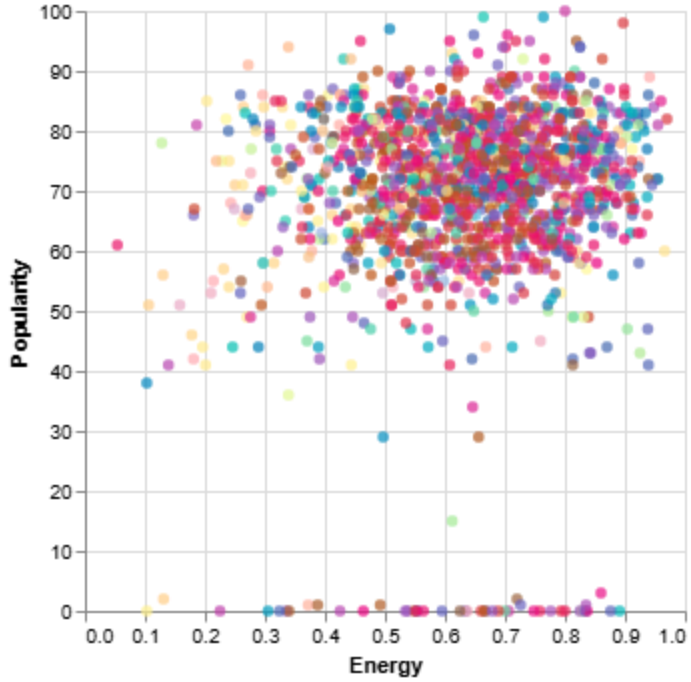
Out[17]:

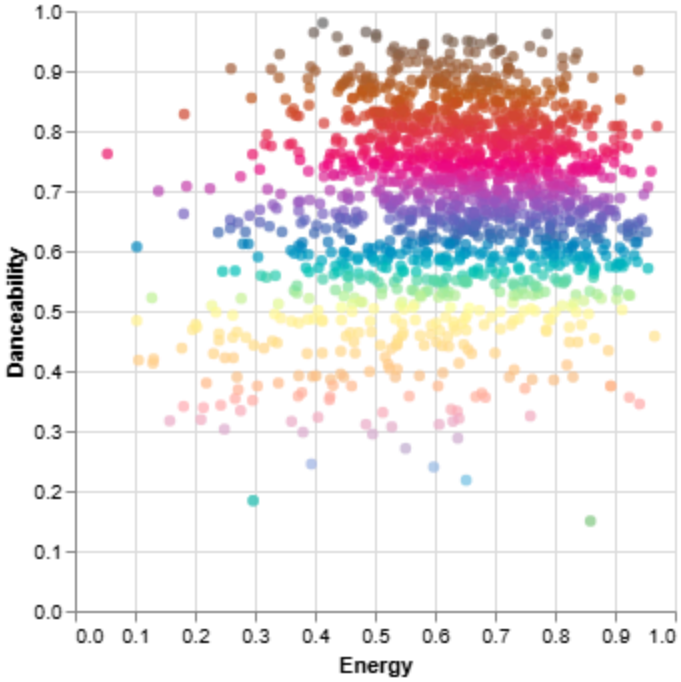

Danceability
0.8
0.6
0.4
0.2
Index
1,600
1,400
1,200
1,000
800
600
400
200
0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy

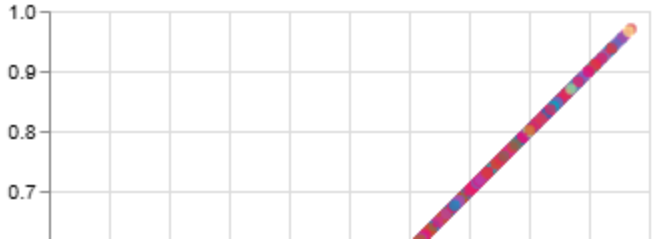

Highest Charting Position
200
180
160
140
120
100
80
60
40
20
0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy

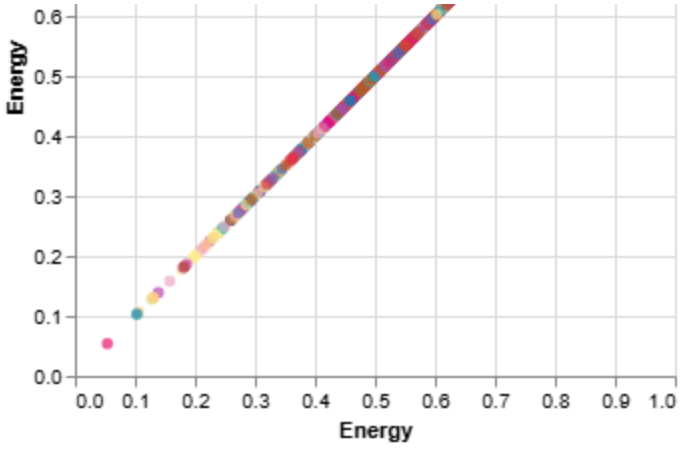

Number of Times Charted
160
140
120
100
80
60
40


20
0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy


Popularity
100
90
80
70
60
50
40
30
20
10
0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy


Danceability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy


1.0
0.9
0.8
0.7

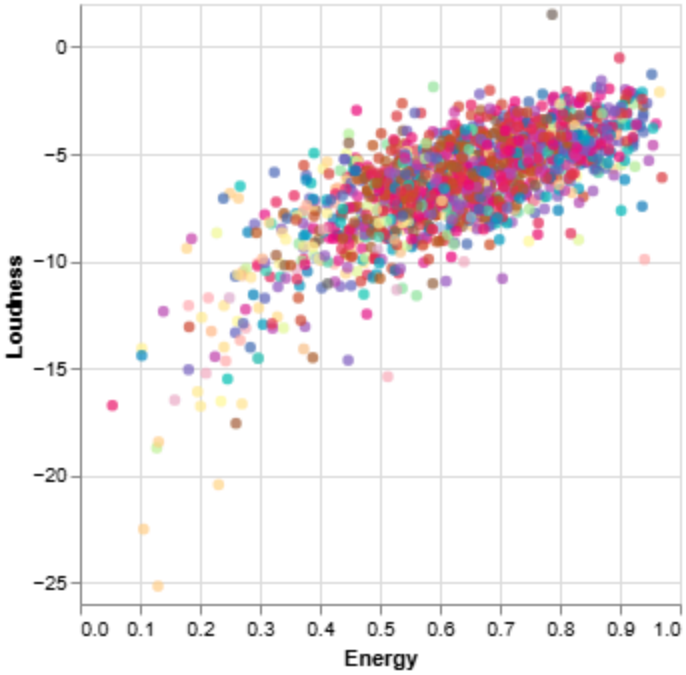


they are good

an in the others?

eft of the

facet chart?

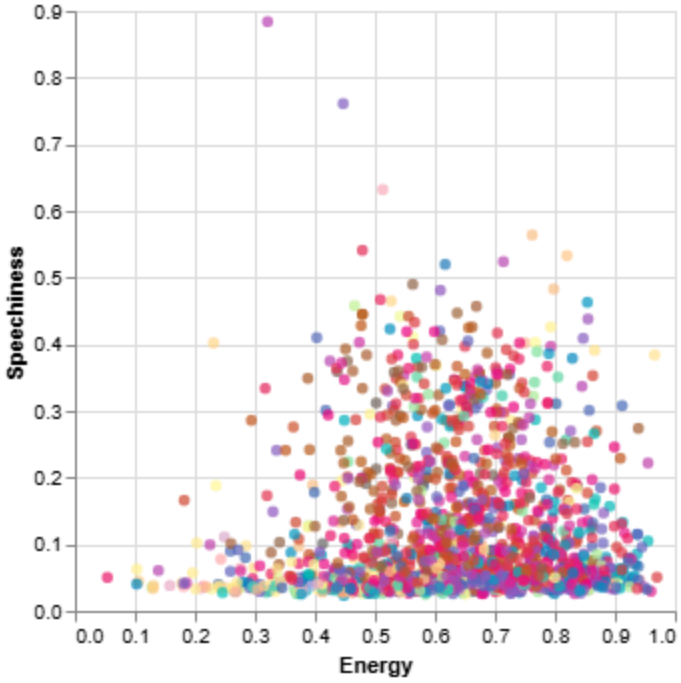


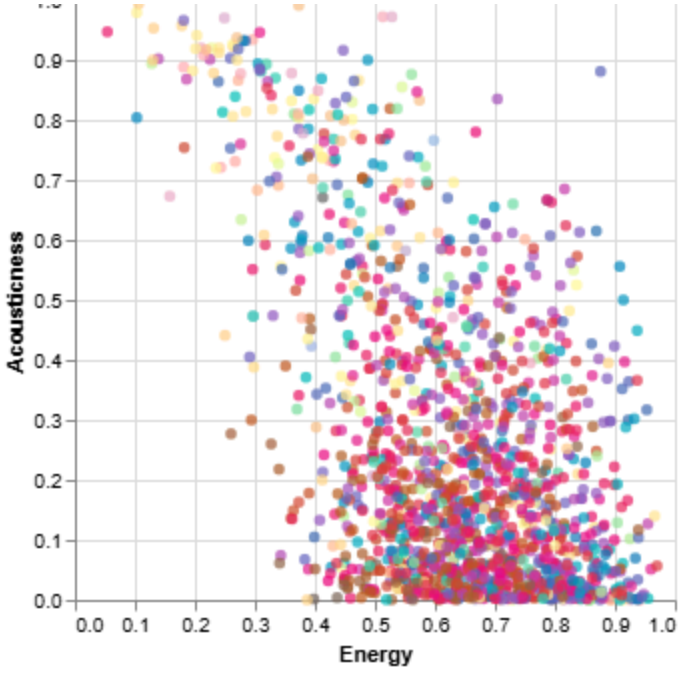
t json file to Canvas.

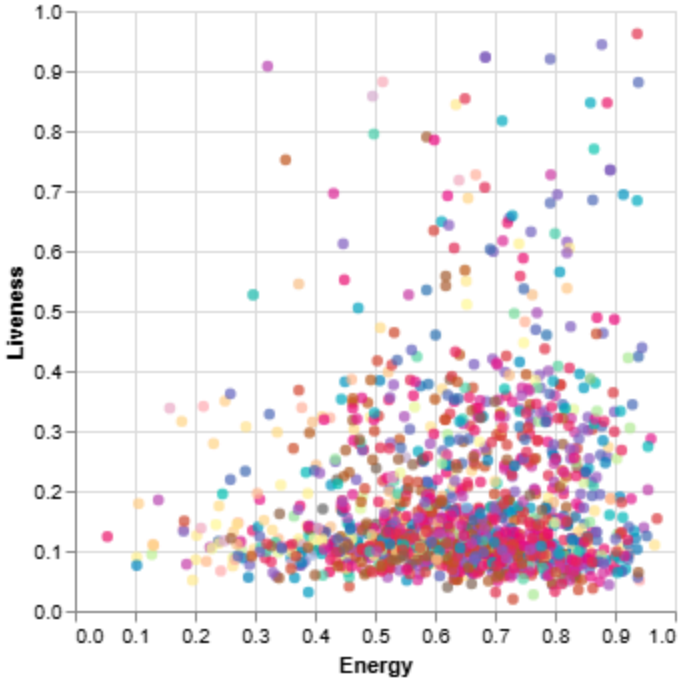
ur computer (it is

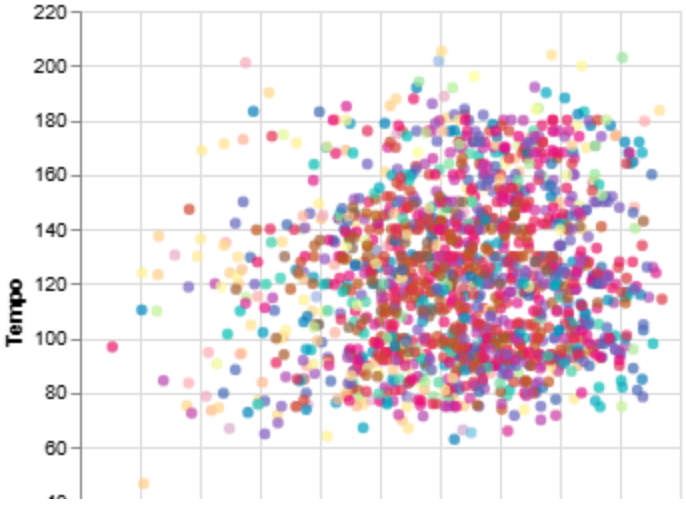
Vega Editor then

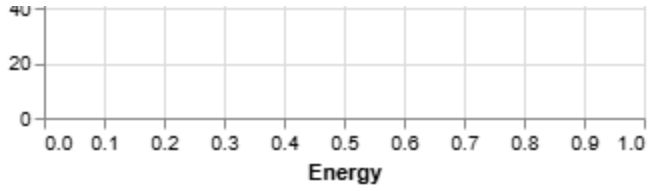
nit the same file).

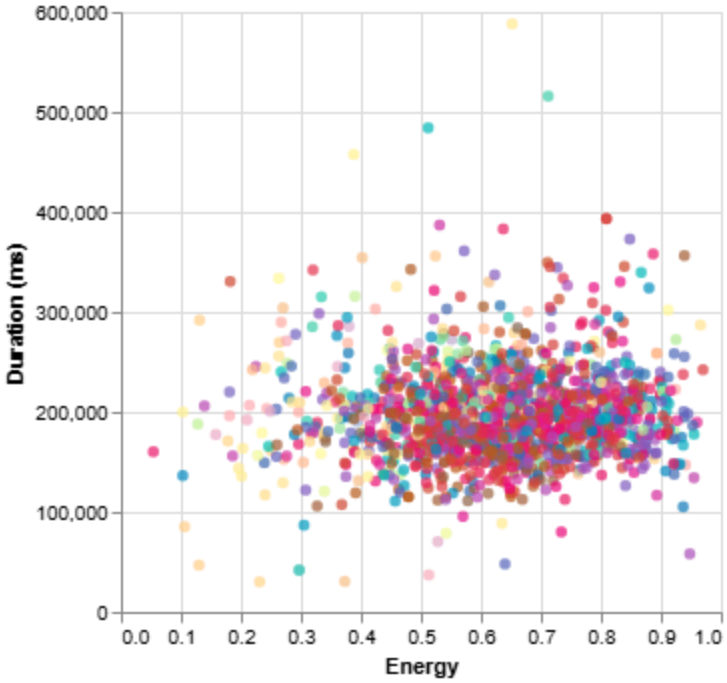


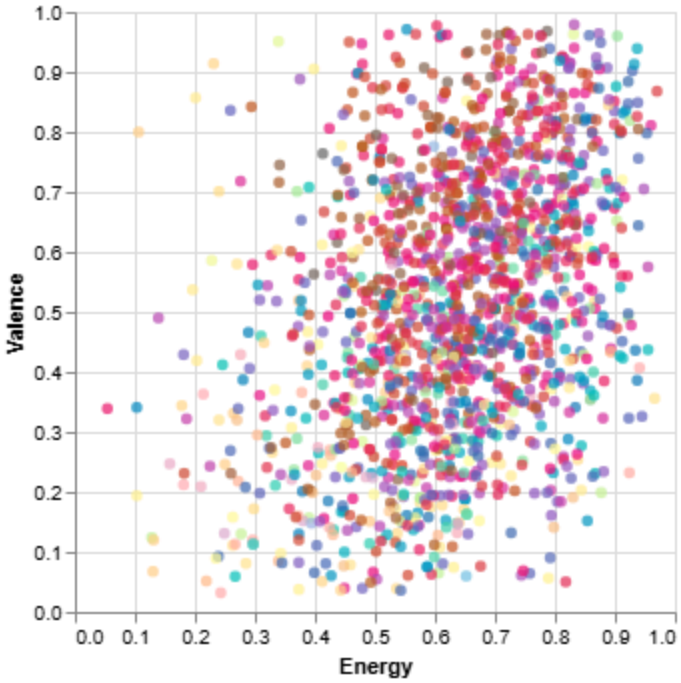

Acousticness
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy


Liveness
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy


Tempo
220 200 180 160 140 120 100 80 60


40
20
0
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy


600,000
500,000
400,000
300,000
200,000
100,000
0
Duration (ms)
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Valence
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
Energy