Customizing with themes

INTRODUCTION TO DATA VISUALIZATION WITH JULIA

Gustavo Vieira Suñe
Data Analyst

datacamp

Plots.jl themes

- Easy customization

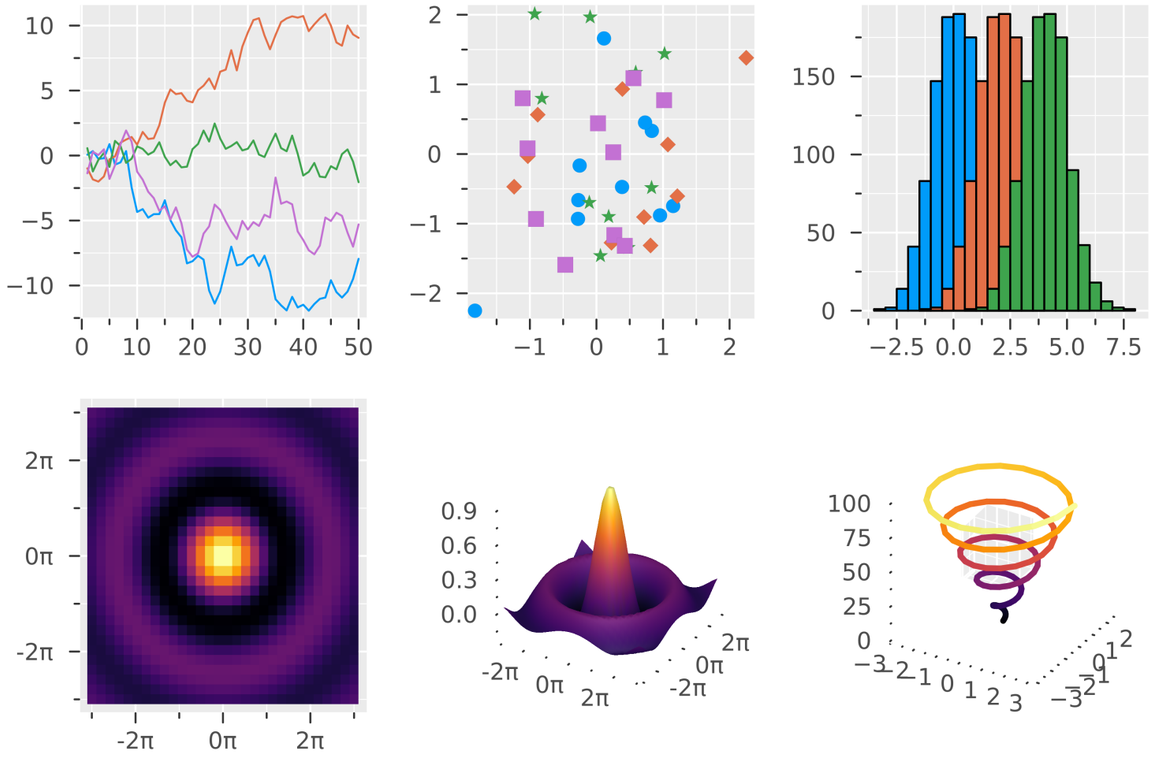
```
using Plots


# Set theme
theme(thm::Symbol; kwargs...)
```

- Many theme (`thm`) choices: `:default` , `:dark` , `:ggplot2` , `:juno` , ...
- `kwargs` : set default attributes

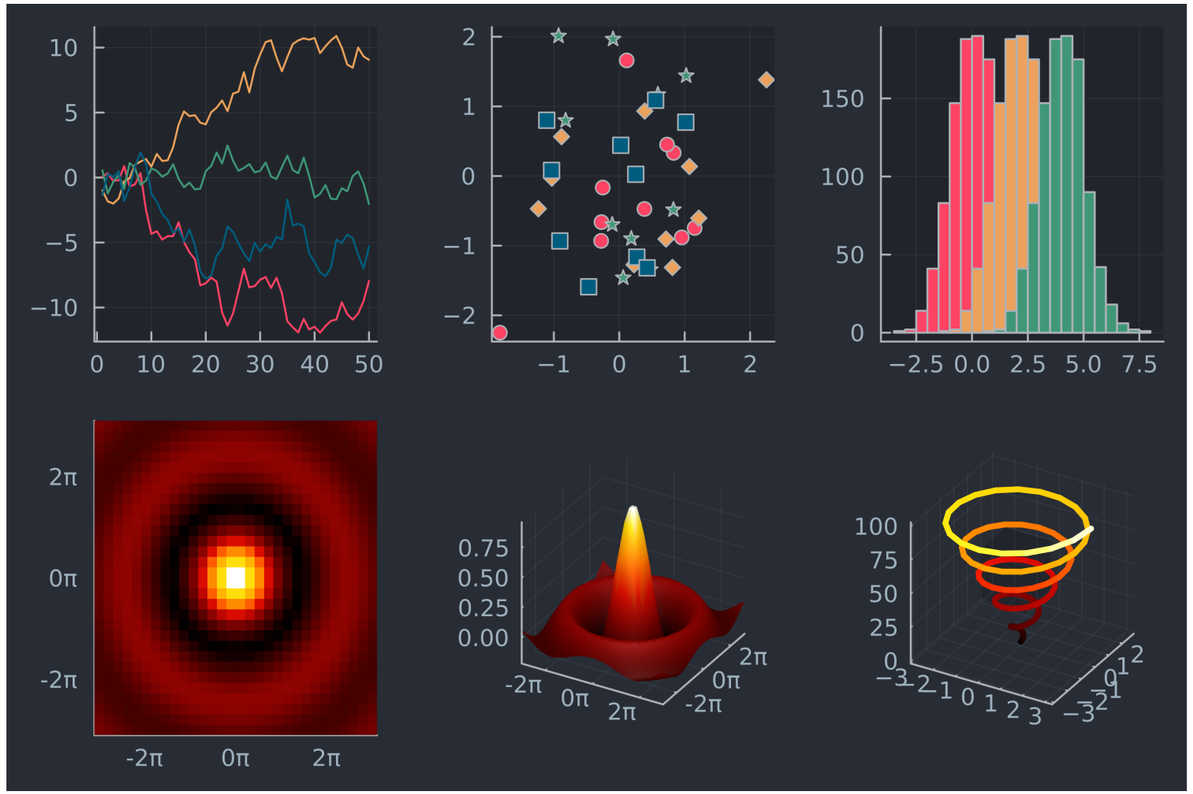
E.g., `theme(:juno, linecolor=:red, linewidth=5)`

Previewing themes

`showtheme(:ggplot2)`



`showtheme(:juno)`



Favorite music genres

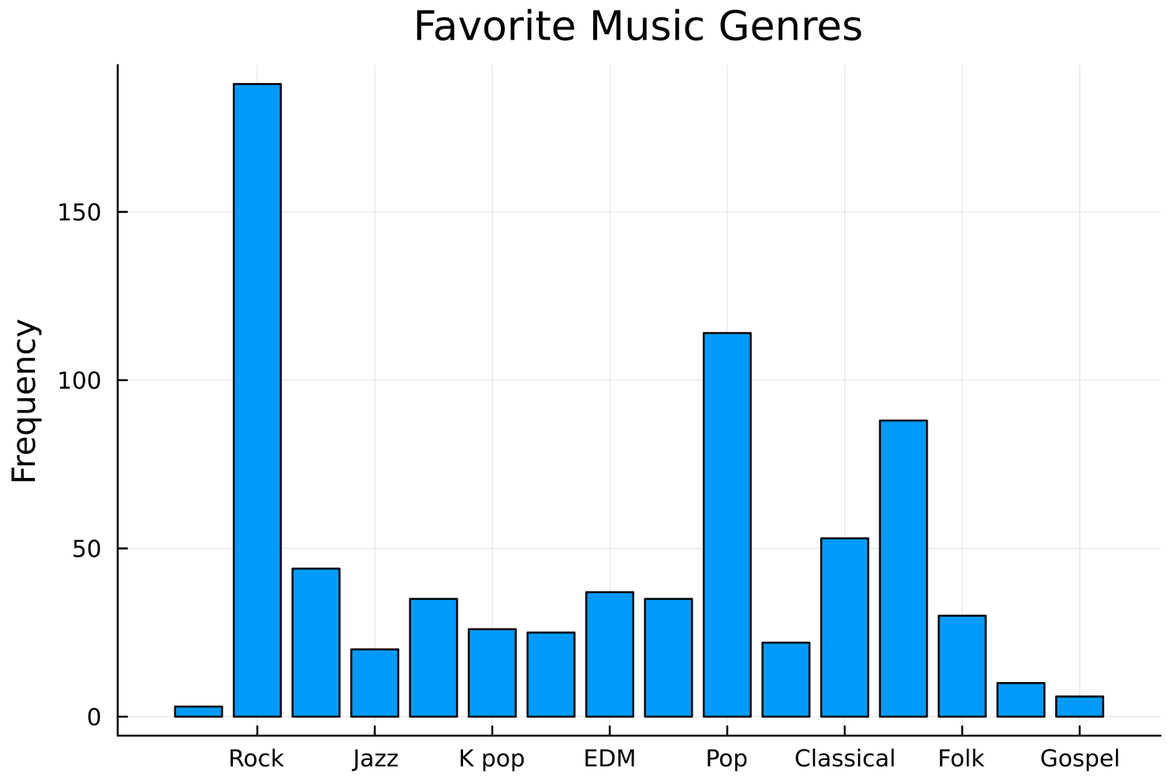
- Music effects on mental health dataset.

```
using DataFrames, CSV, Plots

# Load dataset
streaming = DataFrame(CSV.File("streaming.csv"))

# Group by genre
grouped = groupby(streaming, "Fav genre")
# Count genres
counts = combine(grouped, nrow => :count)
```

- Visualize with bar chart!

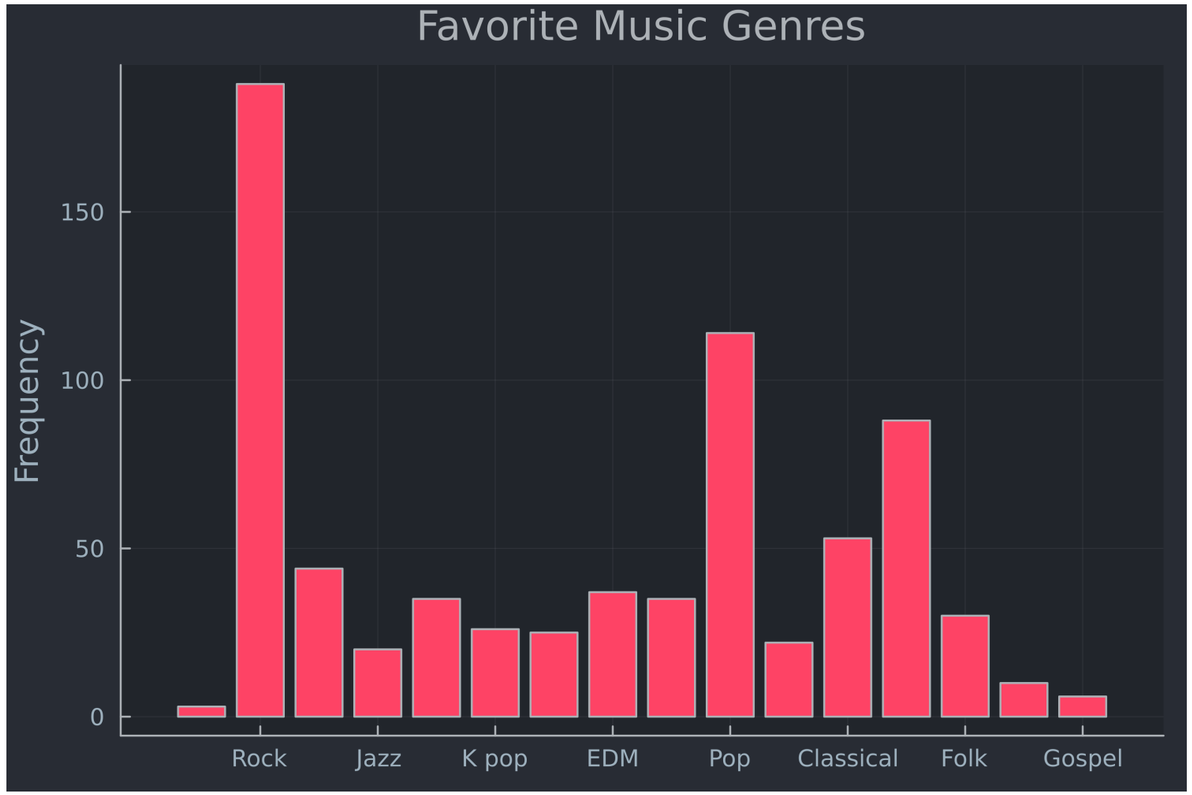
Default-themed bar chart

```
# Bar chart with :default
bar(
    counts[!, "Fav genre"],
    counts.count,
    title="Favorite Music Genres"
    ylabel="Frequency",
    label=false,
)
```



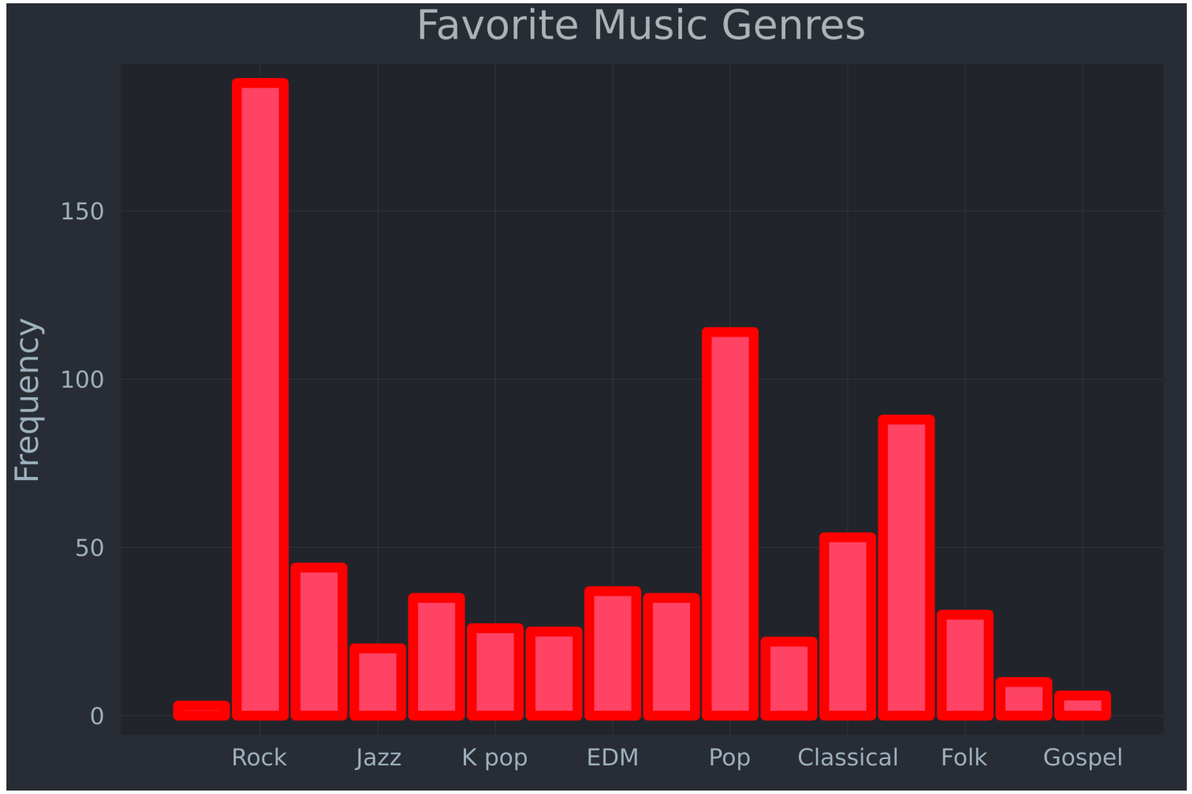
Try a different theme

```
# Set the theme
theme(:juno)

# Bar chart with :juno
bar(
    counts[!, "Fav genre"],
    counts.count,
    label=false,
    ylabel="Frequency",
    title="Favorite Music Genres"
)
```



Make it pop!

```
theme(
    :juno,
    # Default line color and width
    linecolor=:red, linewidth=5,
    # Hide the axis lines
    framestyle=:grid
)
bar(
    counts[!, "Fav genre"],
    counts.count,
    label=false,
    ylabel="Frequency",
    title="Favorite Music Genres"
)
```



- Save to file: `savefig("genres.png")`

Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH JULIA

Plot Attributes and Color Palettes

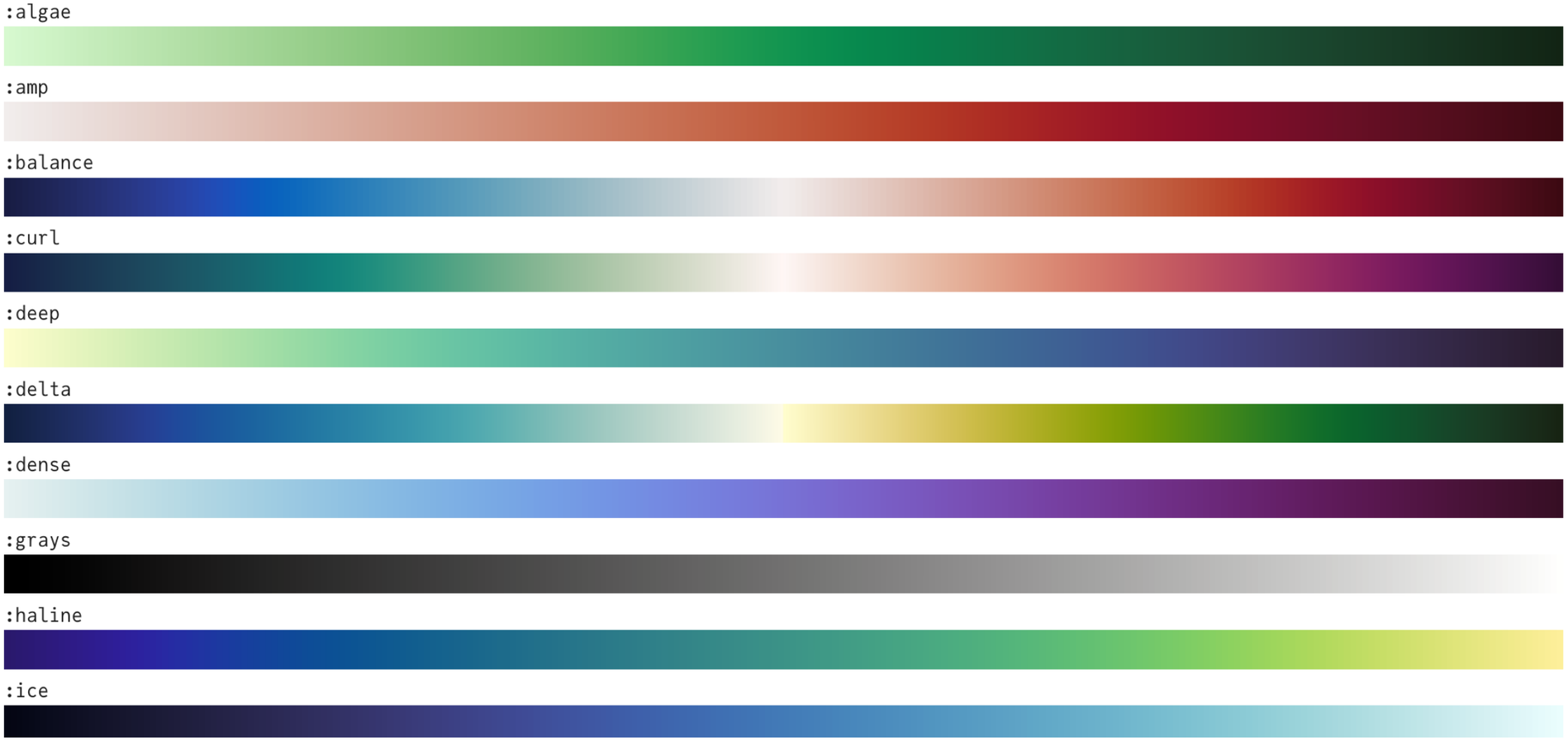
INTRODUCTION TO DATA VISUALIZATION WITH JULIA

Gustavo Vieira Suñe
Data Analyst

datacamp

Color palettes

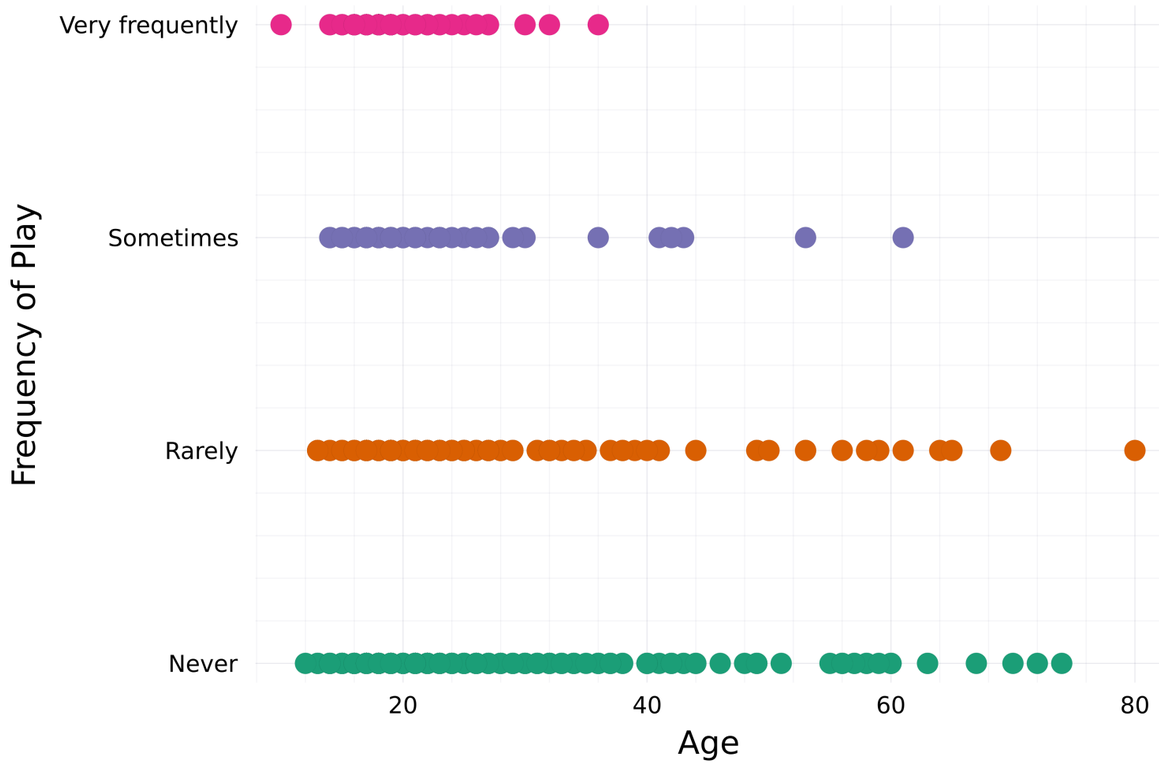
- Many color palettes, including



[1] https://juliagraphics.github.io/ColorSchemes.jl/stable/catalogue/

Using palettes

```
# Set the theme
theme(:bright)

# Create a scatter plot
scatter(
    streaming.Age,
    streaming."Frequency [K pop]",
    group=streaming."Frequency [K pop]",
    label=false,
    # Set color palette
    palette=:Dark2_5
)
xlabel!("Age")
ylabel!("Frequency of Play")
```



Plot attributes

Previously encountered:

- `title`, `xlabel`, `ylabel`
- `xticks`, `yticks`
- `color`
- `linecolor`, `linewidth`
- `markercolor`
- `framestyle`

Many more:

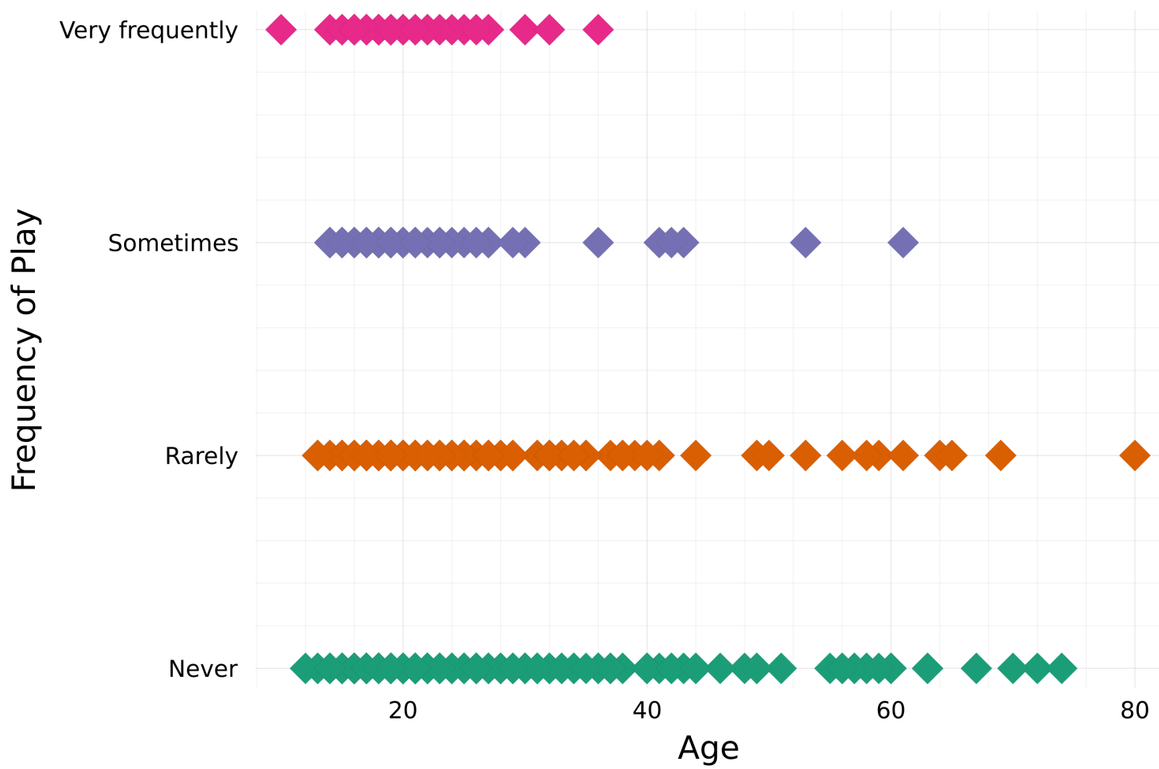
- `xlims`, `ylims`
- `alpha`
- `linestyle`
- `legend_title`, `legend_position`
- `markersize`, `markershape`

...

[1] https://docs.juliaplots.org/stable/attributes/#attributes

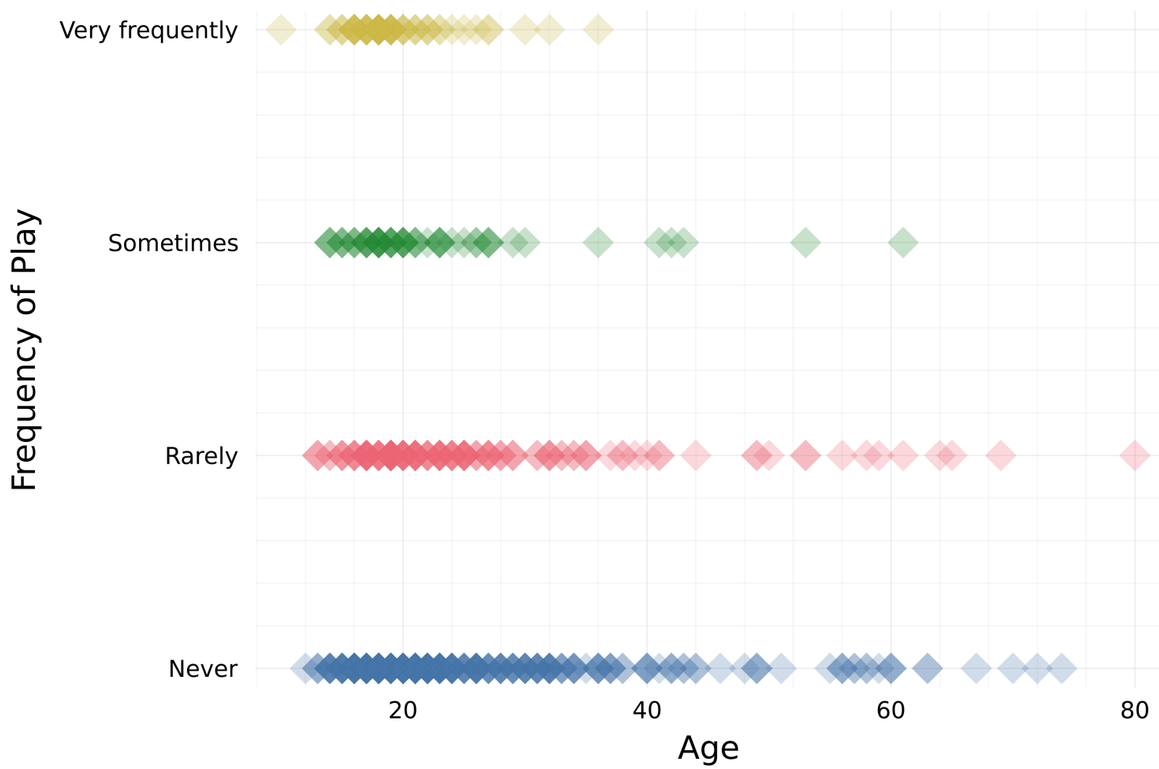
Marker attributes

```
theme(:bright)

scatter(
    streaming.Age,
    streaming."Frequency [K pop]",
    group=streaming."Frequency [K pop]",
    label=false,
    palette=:Dark2_5
    # Marker attributes
    markershape=:diamond,
    markersize=8,
)
xlabel!("Age")
ylabel!("Frequency of Play")
```



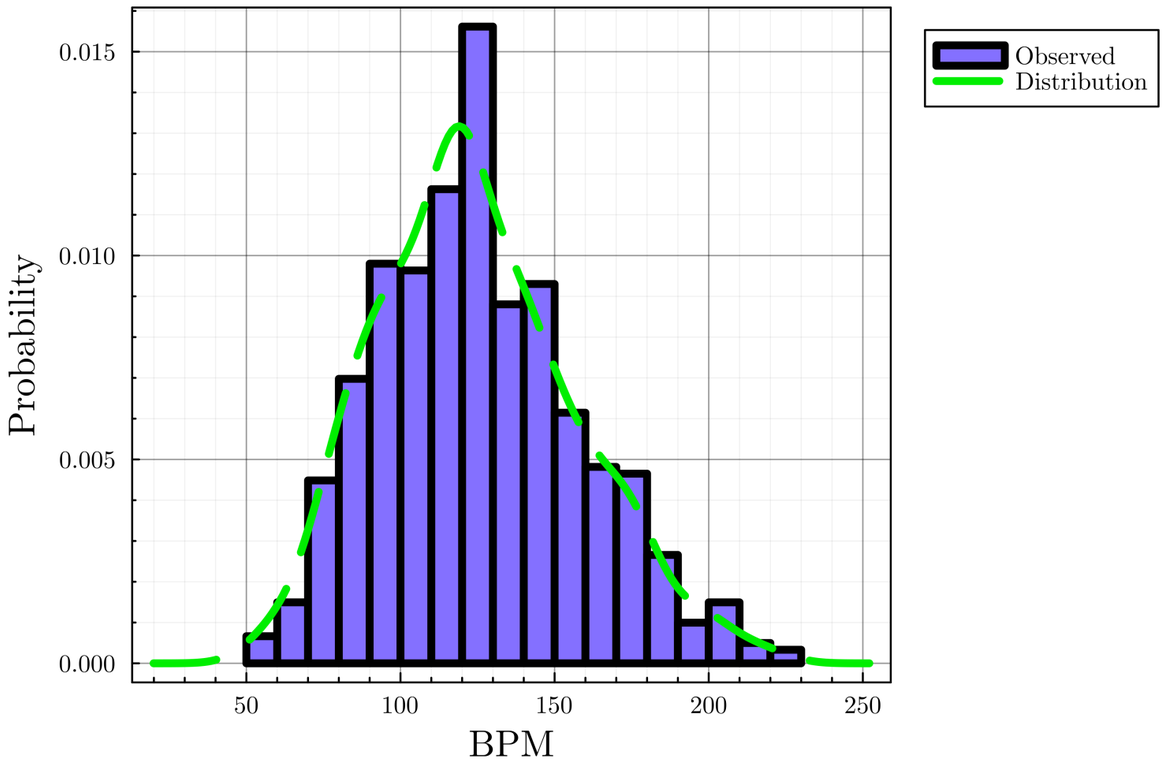
Opacity/Transparency

```
theme(:bright)

scatter(
    streaming.Age,
    streaming."Frequency [K pop]",
    group=streaming."Frequency [K pop]",
    label=false,
    palette=:Dark2_5
    markershape=:diamond,
    markersize=8,
    # Opacity
    alpha=0.25
)
xlabel!("Age")
ylabel!("Frequency of Play")
```



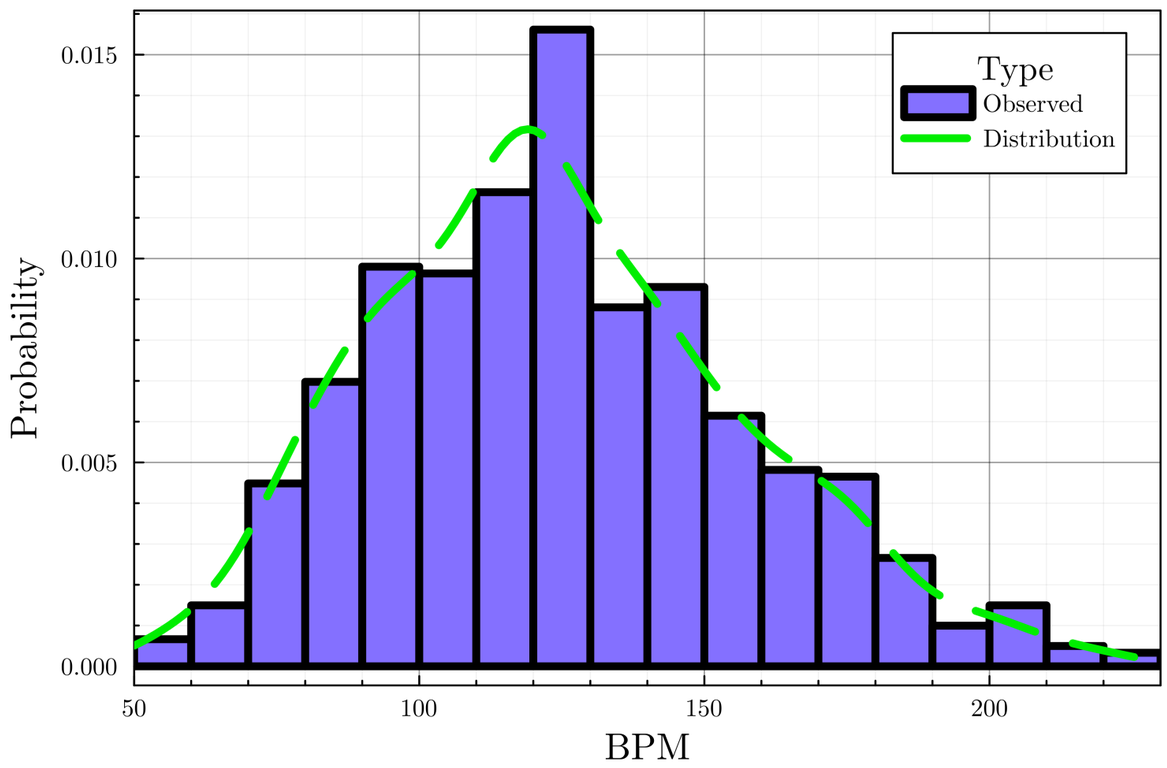
Line attributes

```
# Set theme and default line width
theme(:dao, linewidth=4)
# Create histogram
histogram(streaming.BPM, label="Observed",
    color=:lightslateblue, normalize=true)
# Add density plot
density!(streaming.BPM,
    label="Distribution", linecolor=:green2,
    # Line style
    linestyle=:dash)
xlabel!("BPM")
ylabel!("Probability")
```



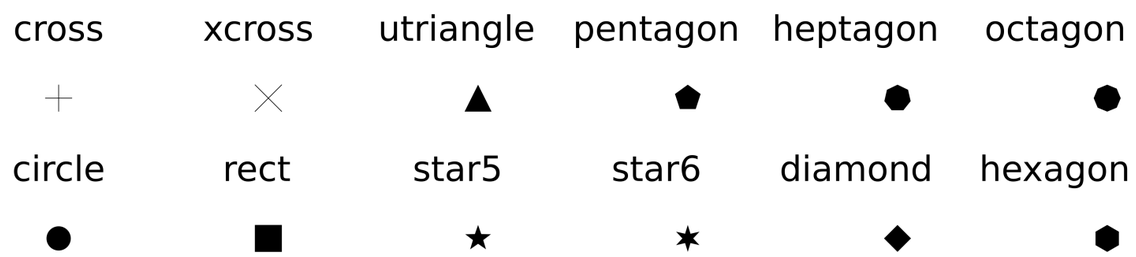
Axis bounds and legend attributes

```
theme(:dao, linewidth=4)

histogram(streaming.BPM, label="Observed",
    color=:lightslateblue, normalize=true)
density!(streaming.BPM, label="Distribution",
    linecolor=:green2, linestyle=:dash
    # Customize legend
    legend_title="Type",
    legend_position=:topright)
# Set the x-axis bounds
xlims!(50, 230)
xlabel!("BPM")
ylabel!("Probability")
```



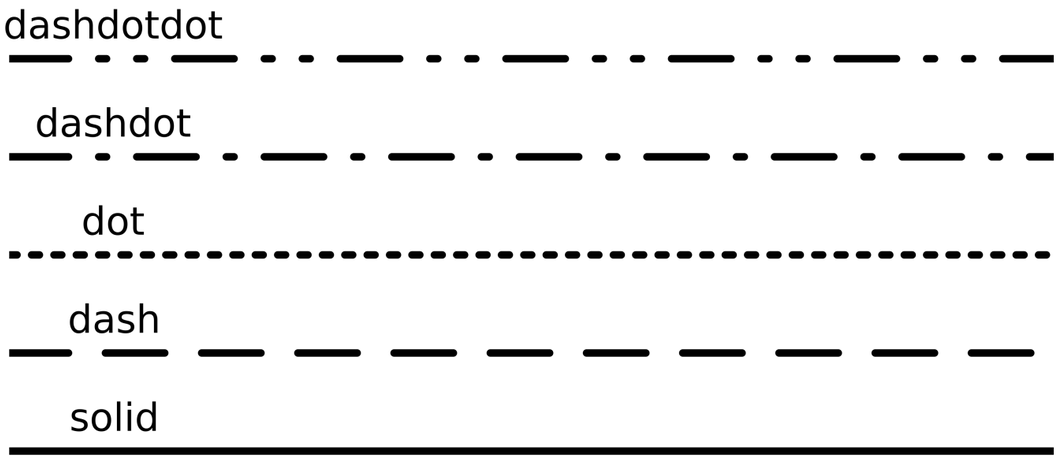
Cheat sheet

- Color palettes
  - `plot(..., palette=palette_symbol)`
- Marker attributes
  - `markersize`
  - `markershape`



- Opacity: `alpha=opacity_value`
- Axis bounds: `xlims!()`, `ylims!()`

- Line attribute: `linetyle`



- Legend attributes
  - `legend_title`
  - `legend_position` (`:right`, `:left`, `:top`, `:bottom`, `:topright`, `:topleft`, ...)

Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH JULIA

Efficient visualizations with layouts

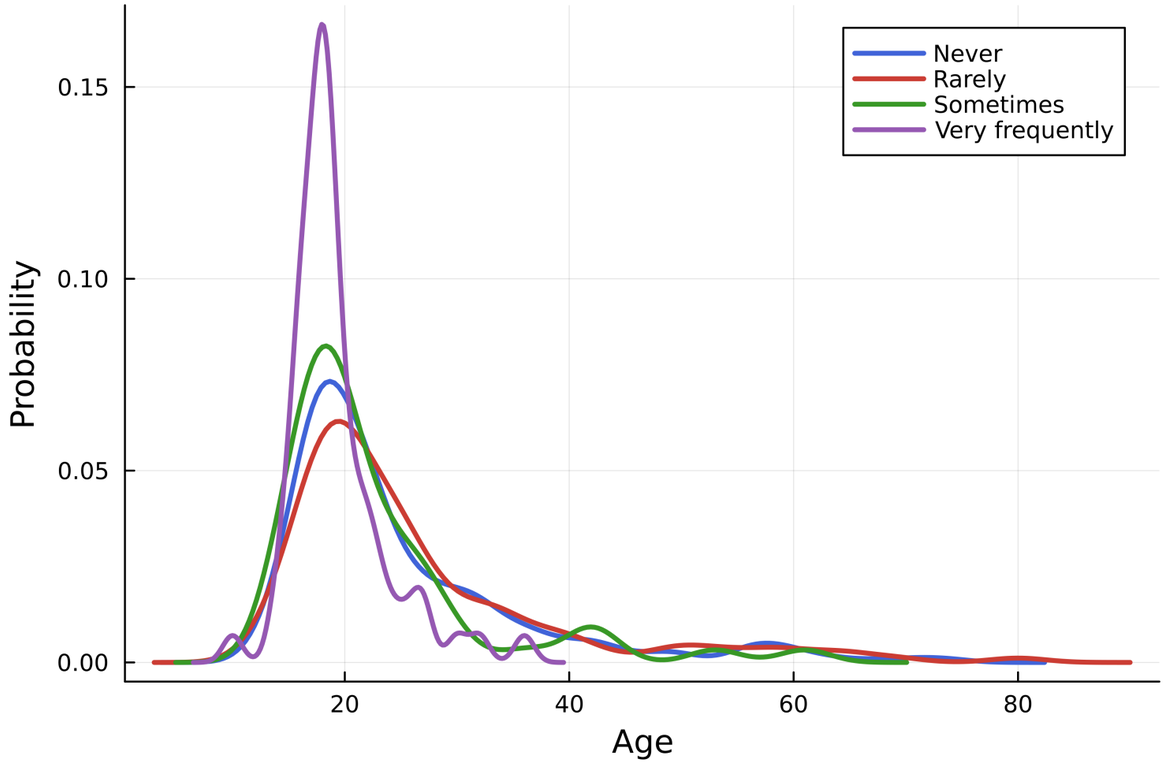
INTRODUCTION TO DATA VISUALIZATION WITH JULIA

Gustavo Vieira Suñe
Data Analyst

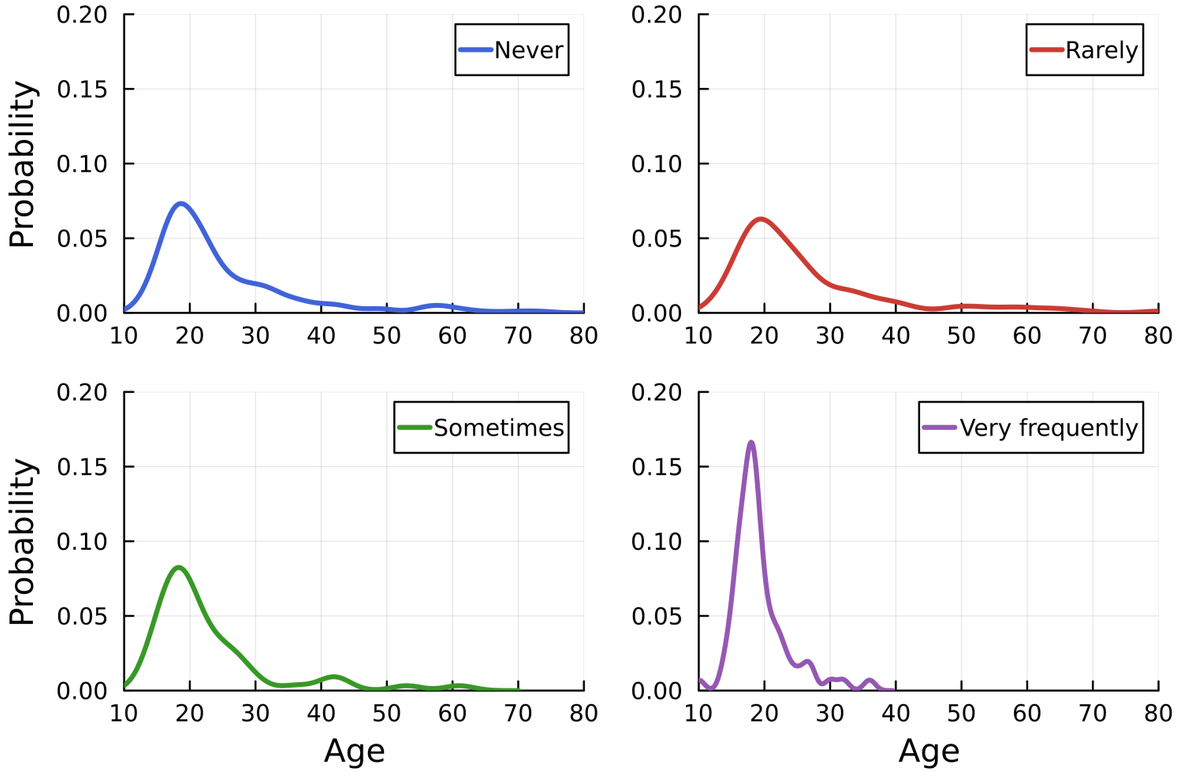
datacamp

Layouts

- Multiple curves in one figure

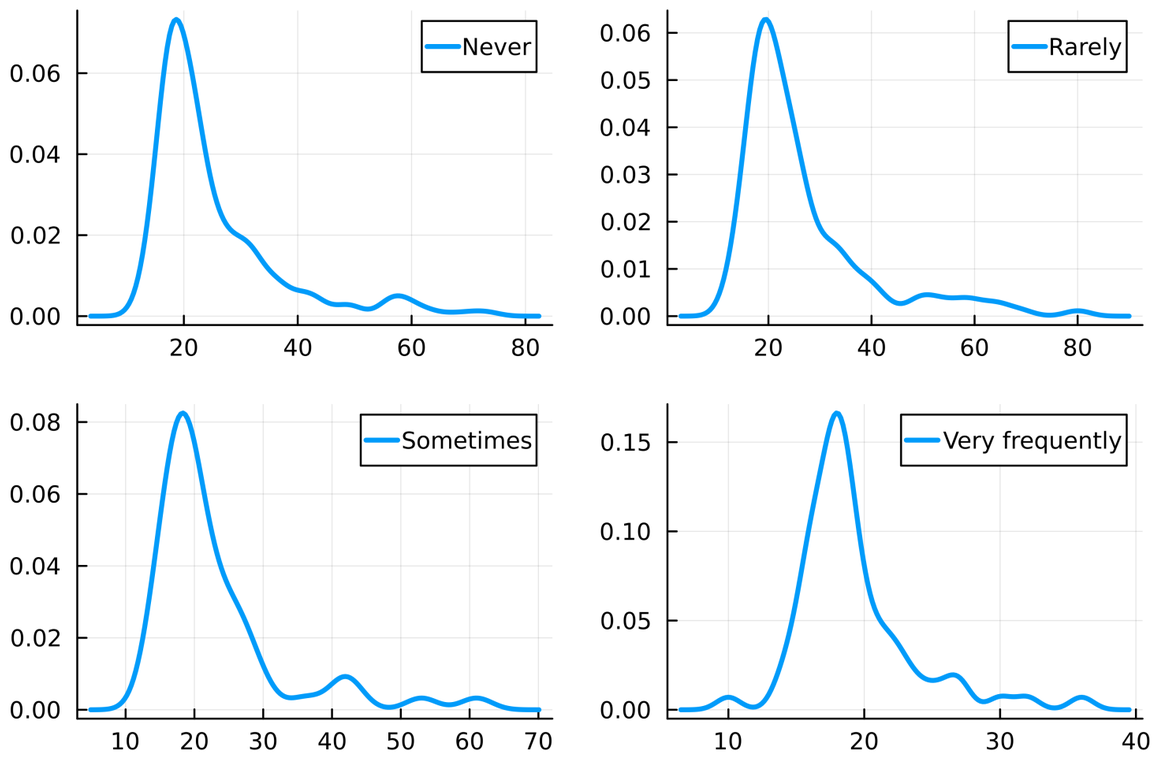


- Plot grid (`layout`)



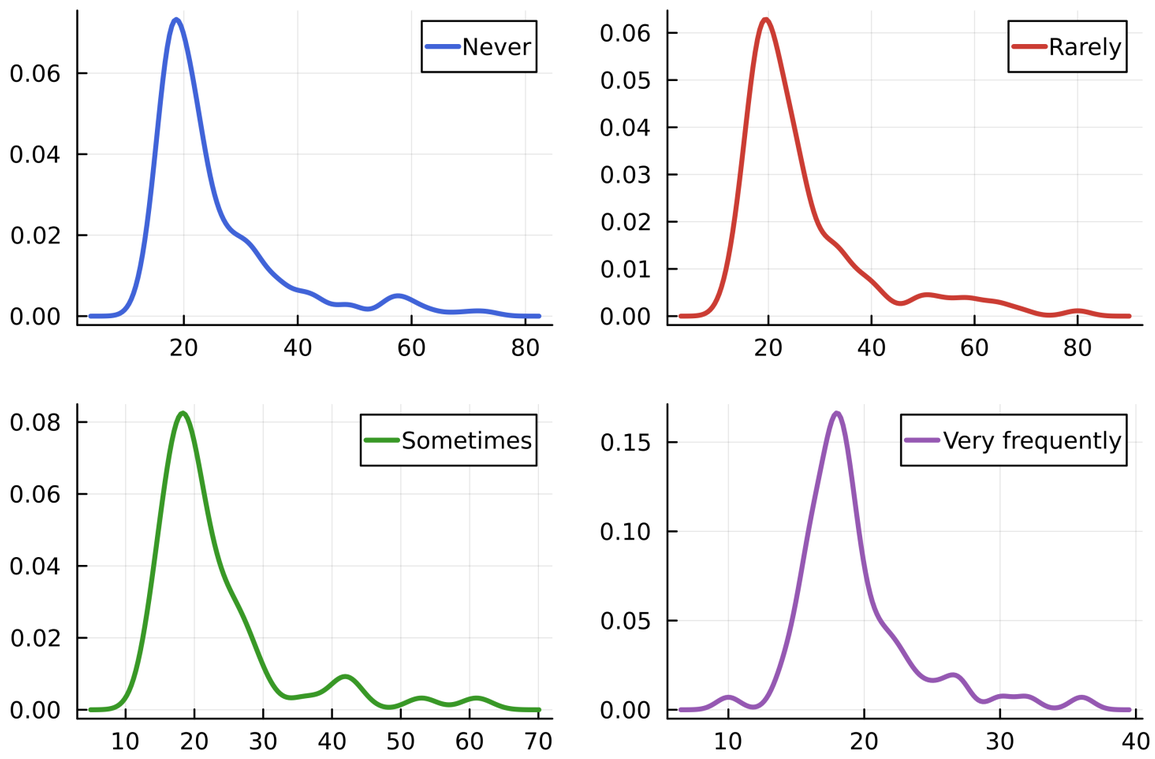
The grid

```
using StatsPlots, DataFrames, CSV

# Load dataset
streaming = DataFrame(
    CSV.File("streaming.csv")
)
# Create density plot
density(
    streaming.Age,
    group=streaming."Frequency [K pop]",
    linewidth=2.5,
    # Set layout
    layout=4,
)
```



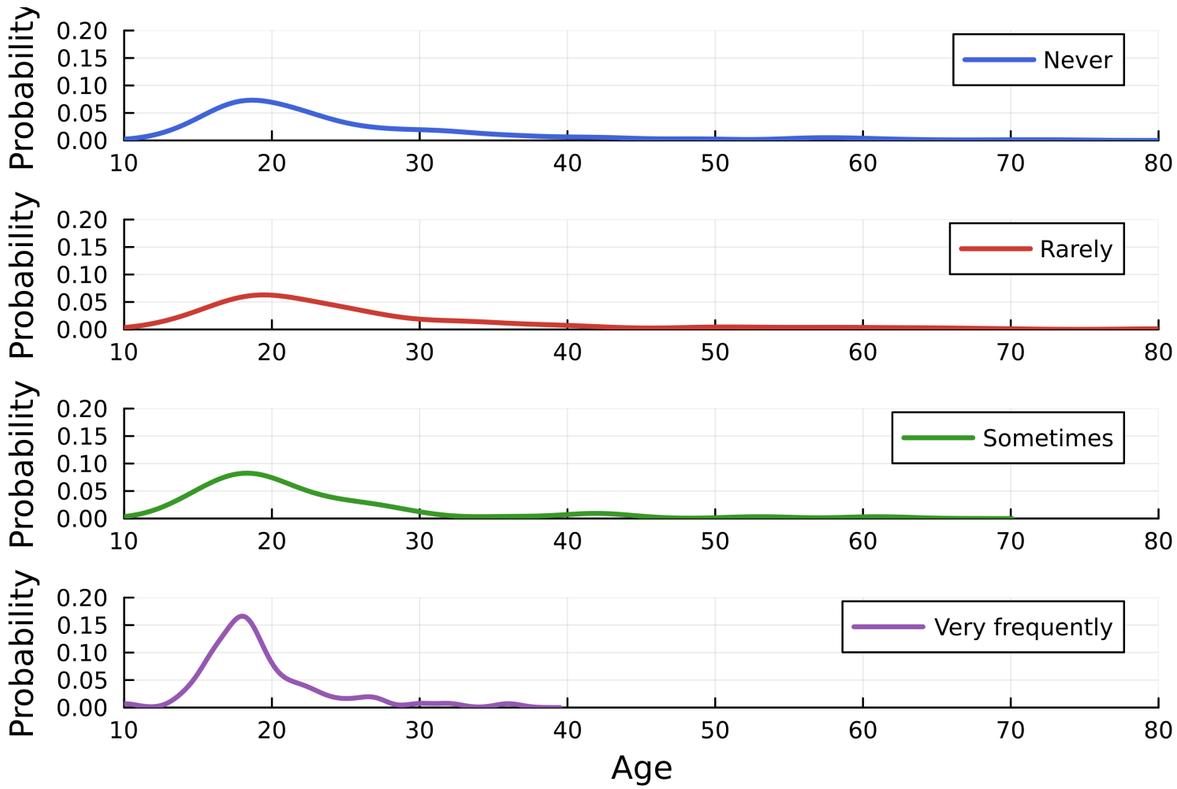
Customizing grid elements

```
using Colors

# Julia logo colors
logocolors = Colors.JULIA_LOGO_COLORS
colors = [logocolors.blue logocolors.red
    logocolors.green logocolors.purple]
density(
    streaming.Age,
    group=streaming."Frequency [K pop]",
    linewidth=2.5,
    layout=4,
    # Line colors
    linecolor=colors,
)
```

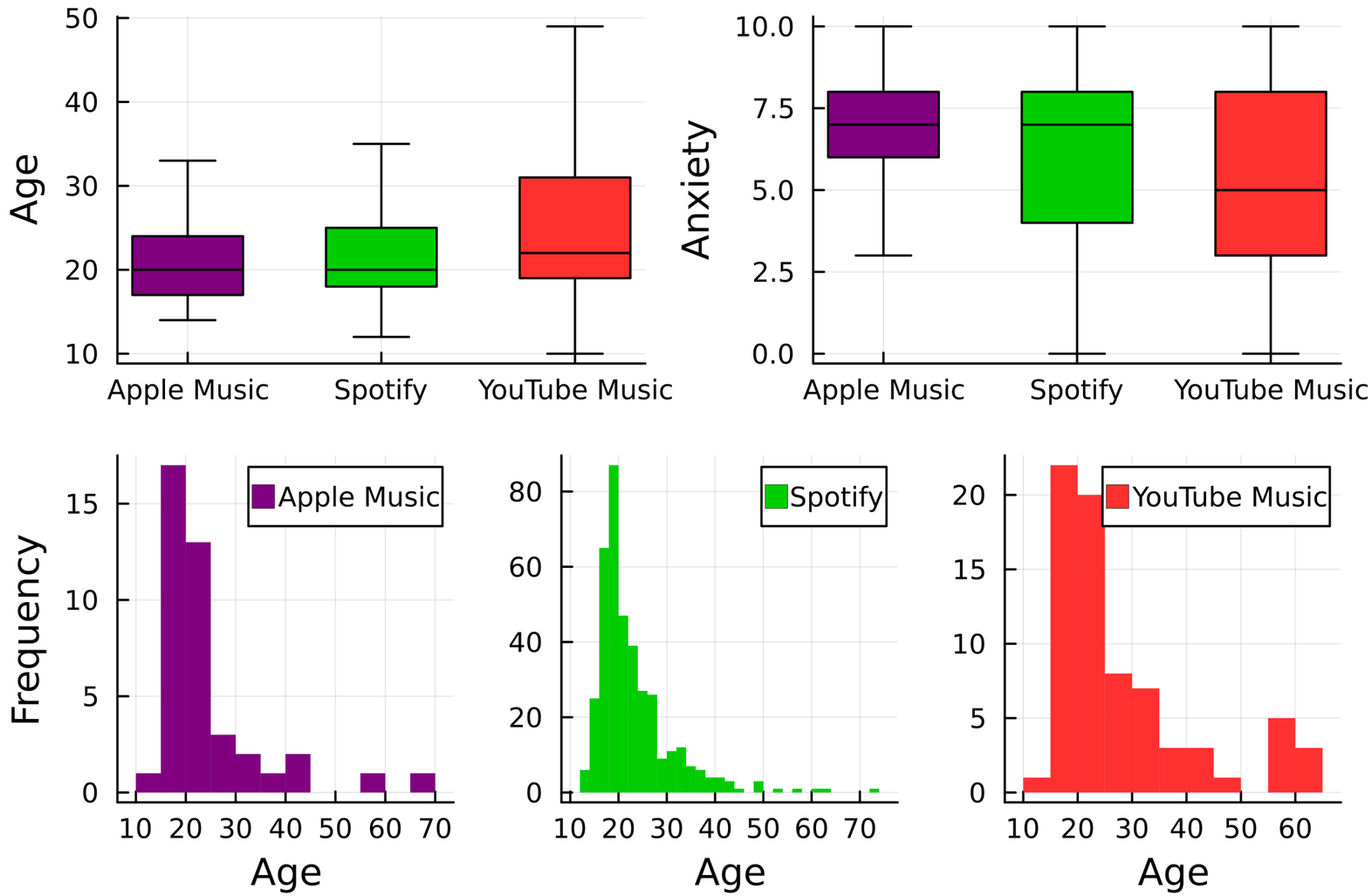


Controlling the grid layout

```
density(
    streaming.Age,
    group=streaming."Frequency [K pop]",
    linewidth=2.5,
    linecolor=colors,
    # Layout dimensions
    layout=(4, 1),
    # Axis labels
    xlabel=["" "" "" "Age"],
    ylabel="Probability",
)
# Axis bounds
xlims!(10, 80)
ylims!(0, 0.2)
```

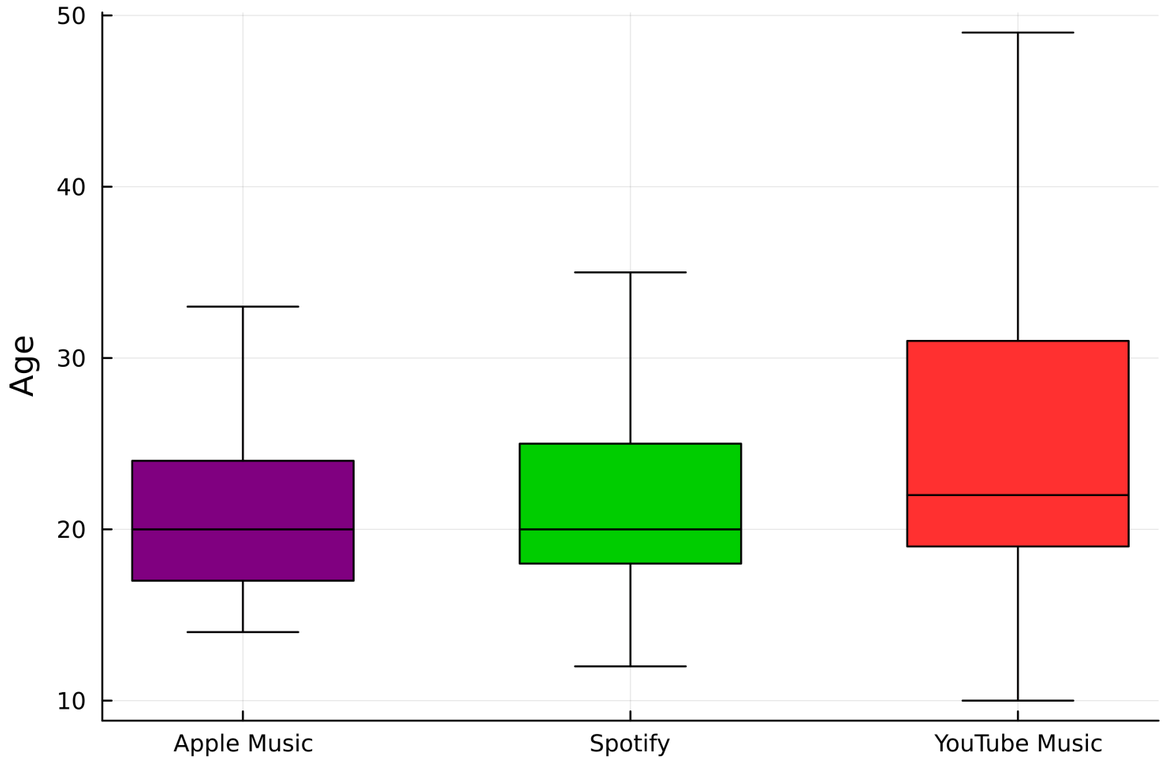


Advanced layouts



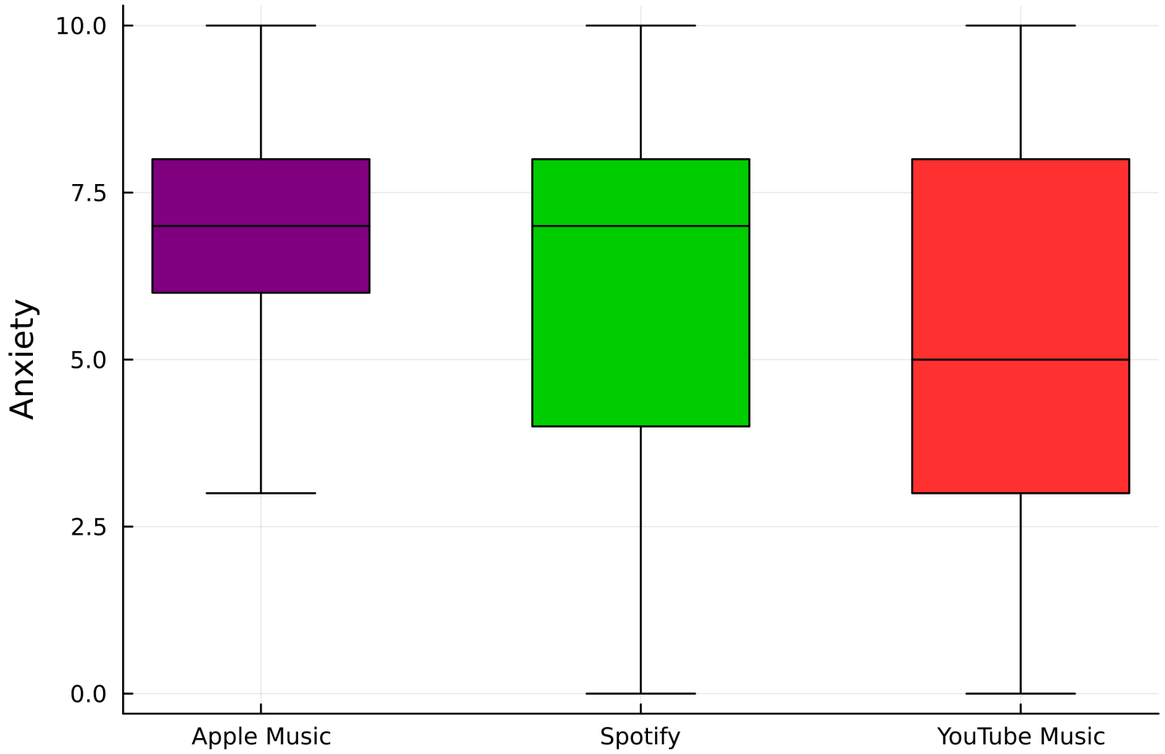
Step-by-step

```
theme(:wong)
# Choose colors
colors = [:purple :green3 :firebrick1]

# First box plot
p1 = boxplot(streaming."Streaming service",
    streaming.Age,
    # Group by streaming service
    group=streaming."Streaming service",
    color=colors,
    label=false,
    ylabel="Age",
    # Remove outliers
    outliers=false)
```



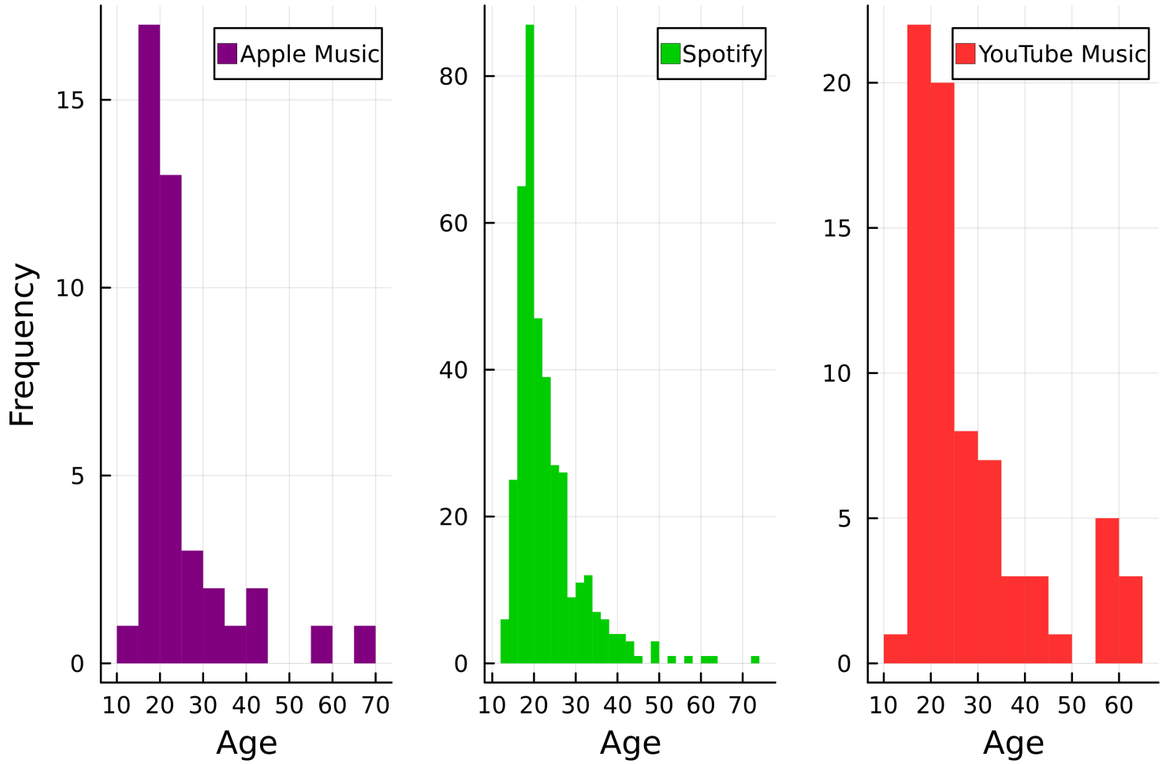
Step-by-step

```
# Second box plot
p2 = boxplot(
    streaming."Streaming service",
    streaming.Anxiety,
    # Group by streaming service
    group=streaming."Streaming service",
    color=colors,
    label=false,
    ylabel="Anxiety",
    # Remove outliers
    outliers=false,
)
```



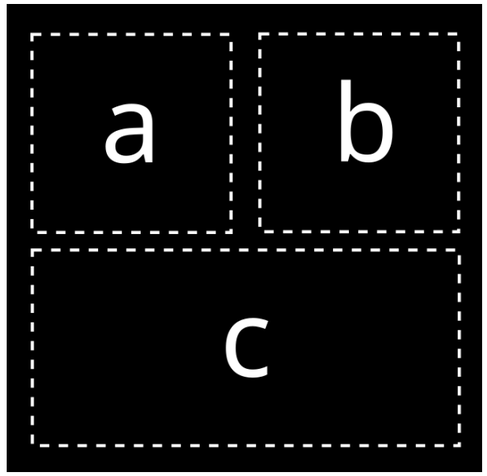
Step-by-step

```
# Histograms
p3 = histogram(
    streaming.Age,
    group=streaming."Streaming service",
    color=colors,
    # Remove lines
    linewidth=0,
    # Set layout
    layout=(1, 3),
    xlabel="Age",
    # Set y-axis labels
    ylabel=["Frequency" "" ""]
)
```

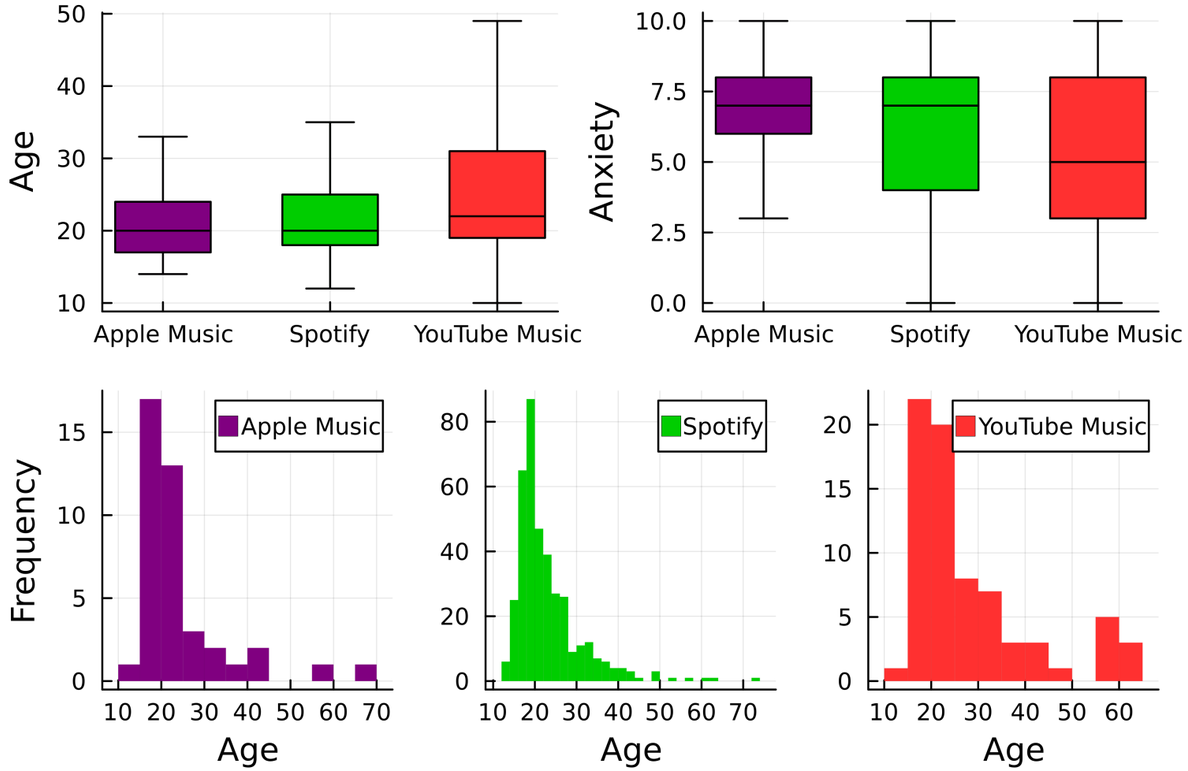


Joining the plots

```
# Select layout
layout = @layout [a b; c]
```




```
# Join the plots
plot(p1, p2, p3, layout=layout)
```



Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH JULIA

Series Recipes

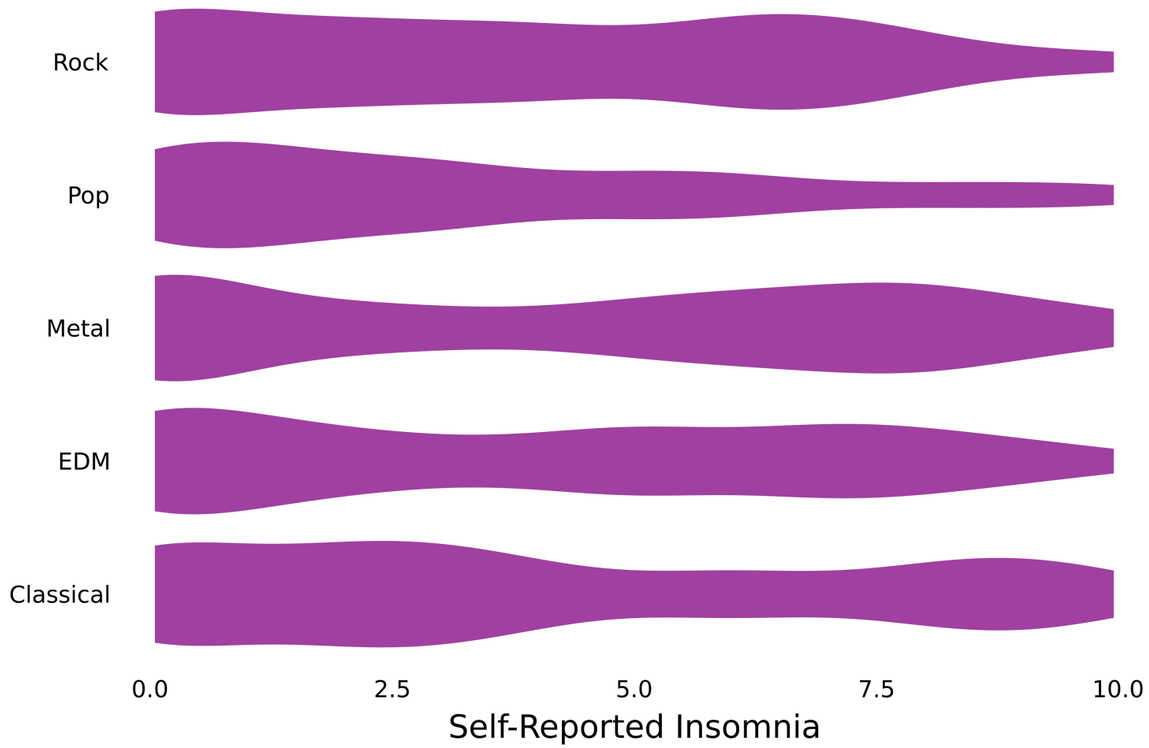
INTRODUCTION TO DATA VISUALIZATION WITH JULIA

Gustavo Vieira Suñe
Data Analyst

datacamp

Customizing a plot

```
# Create a violin plot
violin(
    streaming.Fav_genre,
    streaming.Insomnia,
    framestyle=:grid, label=false,
    linewidth=0,
    # Fill properties
    fillcolor=:purple,
    fillalpha=0.75,
    # Permute the axes
    permute=(:x, :y),
    # Remove grid lines
    grid=:off,
)
xlabel!("Self-Reported Insomnia")
```

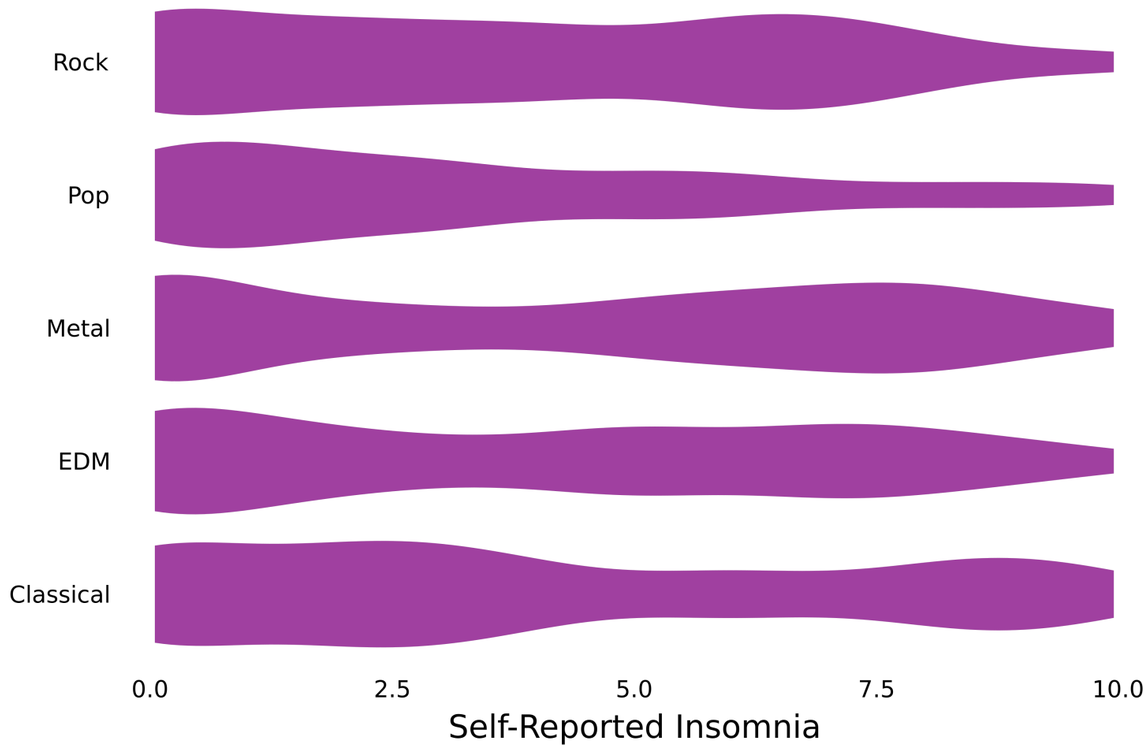


Plots.jl series

- Collection of points sharing the same plotting characteristics
  - `plot(x, y)` points are a series of the `:line` type
- Many series types, including `:line`, `:scatter`, `:histogram`, `:density`, `:bar`
- A plot can have multiple series
  - `scatter(x, [y1 y2])` has two series of type `:scatter`
  - Each column is a series
- Plots.jl follows a different recipe for each series type
  - We can create our own custom recipes!

It's all plot in the end

```
# Create a violin plot
plot(
    streaming.Fav_genre,
    streaming.Insomnia,
    framestyle=:grid, label=false,
    linewidth=0,
    fillcolor=:purple,
    fillalpha=0.75,
    permute=(:x, :y),
    grid=:off,
    # Specify series type
    seriestype=:violin,
)
xlabel!("Self-Reported Insomnia")
```



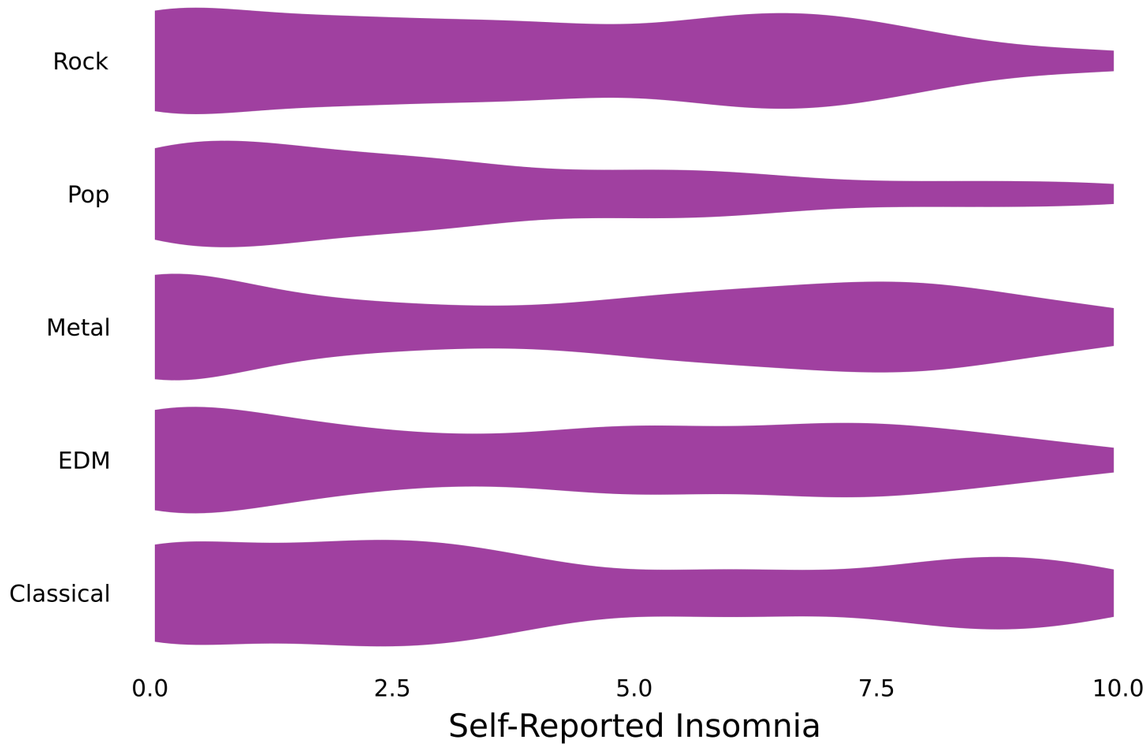
Custom series recipes

```
@recipe function f(
    ::Type{Val{:my_hviolin}}, x, y, z
)
    # Series type
    seriestype := :violin
    # Customization options
    framestyle := :grid
    label := false
    linewidth := 0
    fillcolor := :purple
    fillalpha := 0.75
    permute := (:x, :y)
end
```

- `::Type{Val{:my_hviolin}}` defines `my_hviolin` as the series recipe name
- `x, y, z` represent the series data
- `seriestype` gives the type of the series used in the recipe
- Customization arguments set with `:=`

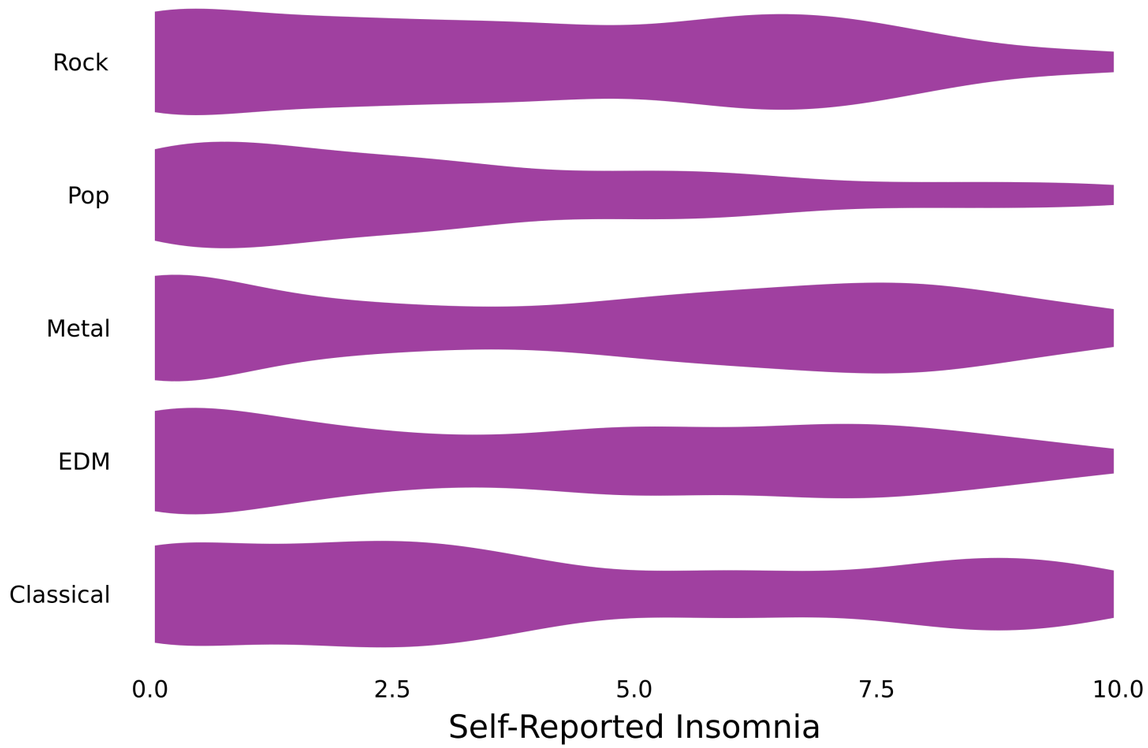
Using custom series recipe

```
# Use recipe
plot(
    streaming.Fav_genre,
    streaming.Insomnia,
    # Specify series recipe
    seriestype=:my_hviolin,
)
xlabel!("Self-Reported Insomnia")
```



Using custom series recipe

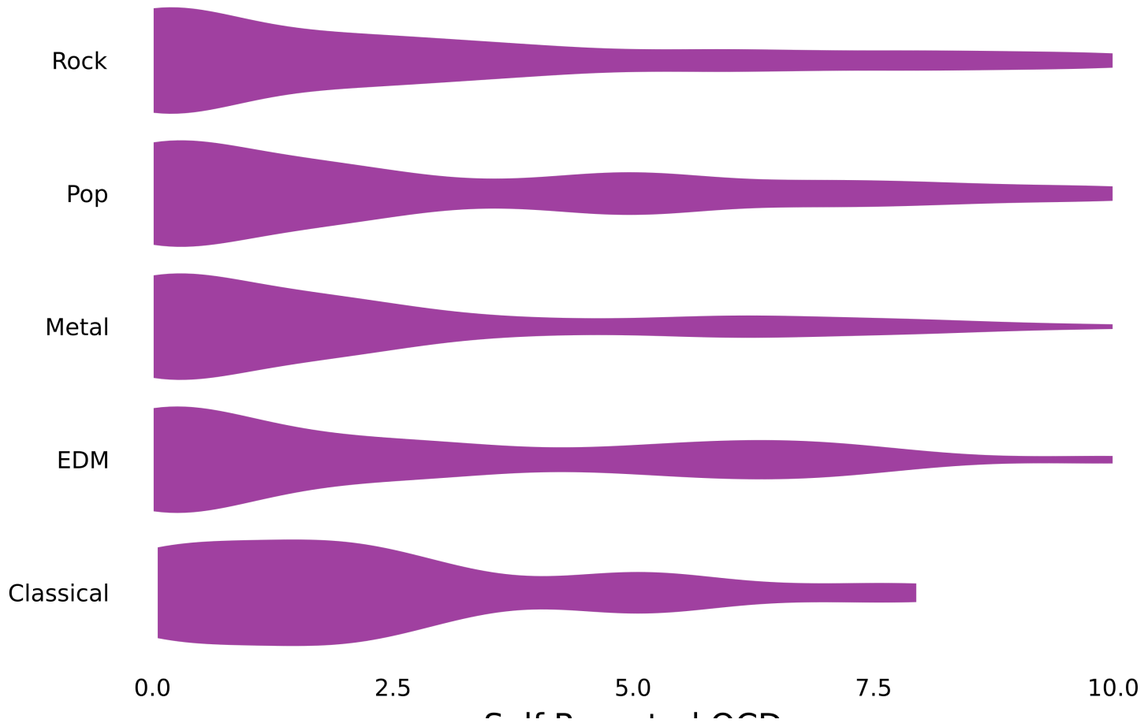
```
# Define my_hviolin function
@shorthands my_hviolin

# Use recipe
my_hviolin(
    streaming.Fav_genre,
    streaming.Insomnia
)
xlabel!("Self-Reported Insomnia")
```

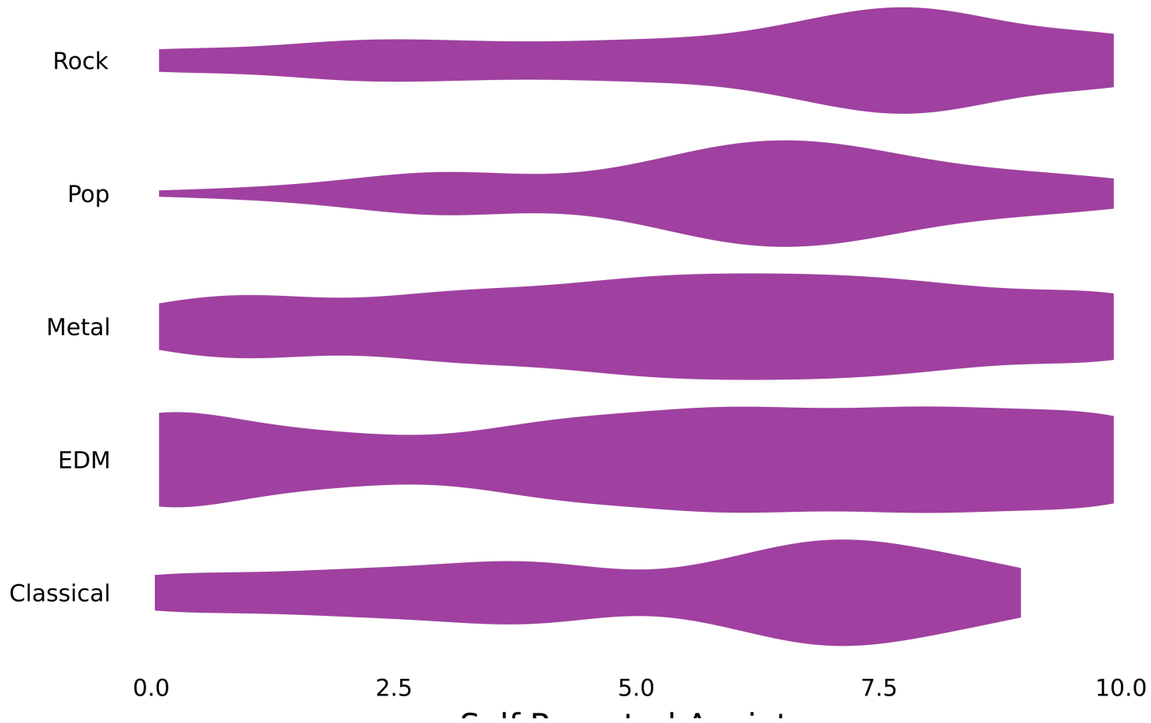


Same recipe, different data

```
my_hviolin(streaming.Fav_genre,
    streaming.OCD)
xlabel!("Self-Reported OCD")
```

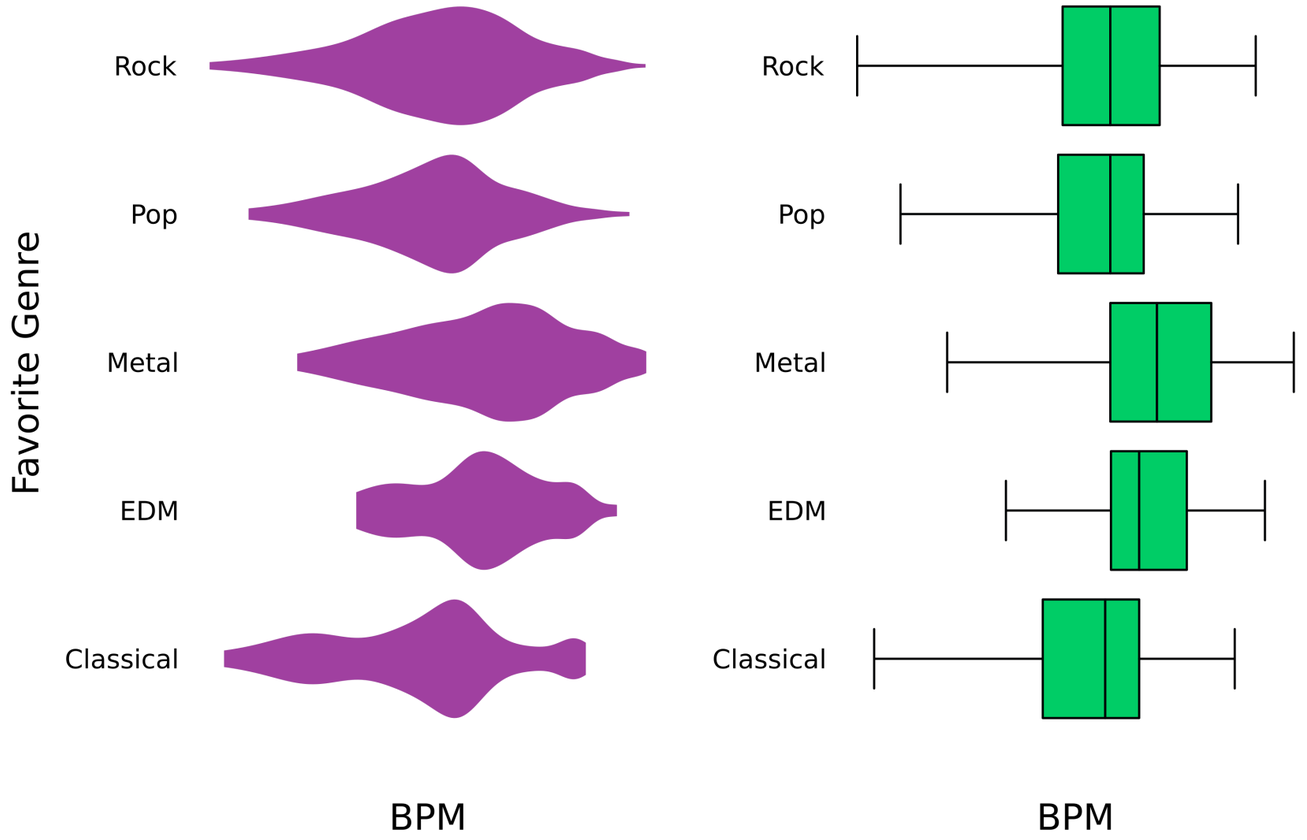
```
my_hviolin(streaming.Fav_genre,
    streaming.Anxiety)
xlabel!("Self-Reported Anxiety")
```





Plot recipes

- Multiple series



[1] https://docs.juliaplots.org/latest/recipes/

Let's practice!

INTRODUCTION TO DATA VISUALIZATION WITH JULIA