PLOTTING WITH JULIA

Isabel Restrepo, PhD | PHP 2561, Brown University | April 25, 2017

Slides: https://gitpitch.com/bcbi/julia_tutorials/master?p=plotting

LIBRARIES

PLOTS.JL

Highlevel, poweful library that wraps many plotting backends

https://juliaplots.github.io/

PROS

- Flexibility - use your favorite backend library to produce your plots
- Consistency - change backend without changing your code
- Smart - use features such as recipes and layouts
- Great Documentation

CONS

- Does not support every possible backend
- In some cases some features may not be implemented

GLADFLY.JL

A Julia implementation inspired by the "Grammar of Graphics" and ggplot2. Primary author: Daniel C Jones
https://github.com/GiovineItalia/Gadfly.jl

PROS:

- Clean look Lots of features
- Flexible when combined with Compose.jl (inset plots, etc)
- Familiar to R users
- Good aesthetics

CONS:

- Does not support 3D
- Slow time-to-first-plot
- Lots of dependencies
- No interactivity

PYPLOT.JL

A Julia wrapper around the popular python package PyPlot (Matplotlib). It uses PyCall.jl to pass data with minimal overhead.
https://github.com/JuliaPy/PyPlot.jl

PROS:

- Tons of functionality
- 2D and 3D
- Mature library
- Standalone or inline
- Well supported in Plots
- Familiar to python users

CONS:

- Uses python
- Dependencies frequently cause setup issues
- Inconsistent output depending on Matplotlib version

PLOTLYJS/PLOTLY

Both libraries have basically identical interface, one uses local resources, the other the cloud. Plotly.js is built on top of d3.js and stack.gl to create a high-level, declarative charting library. plotly.js ships with 20 chart types, including 3D charts, statistical graphs, and SVG maps. PlotlyJS is the corresponding Julia interface. This package constructs plotly graphics using all local resources. To interact or save graphics to the Plotly cloud, use the plotly.jl library.

https://github.com/sglyon/PlotlyJS.jl

PROS:

- Tons of functionality/Super configurable
- 2D and 3D
- Mature library
- Interactivity (even when inline)
- Standalone or inline
- Great looking plots

CONS:

- No custom shapes
- JSON may limit performance

LET'S START WITH PLOTLYJS

Create a Julia notebook called plotlyjs_basics

IMPORT

```
using PlotlyJS
```

```
function linescatter1()
    trace1 = scatter(;x=1:4, y=[10, 15, 13, 17], mode="markers")
    trace2 = scatter(;x=2:5, y=[16, 5, 11, 9], mode="lines")
    trace3 = scatter(;x=1:4, y=[12, 9, 15, 12], mode="lines+markers")
    plot([trace1, trace2, trace3])
end
linescatter1()
```

```
function linescatter3()
    trace1 = scatter(;x=1:5, y=[1, 6, 3, 6, 1],
                      mode="markers+text", name="Team A",
                      textposition="top center",
                      text=["A-1", "A-2", "A-3", "A-4", "A-5"],
                      marker_size=12, textfont_family="Raleway, sans-serif")

    trace2 = scatter(;x=1:5+0.5, y=[4, 1, 7, 1, 4],
                      mode="markers+text", name= "Team B",
                      textposition="bottom center",
                      text= ["B-a", "B-b", "B-c", "B-d", "B-e"],
                      marker_size=12, textfont_family="Times New Roman")

    data = [trace1, trace2]
```

```
function linescatter5()

    country = ["Switzerland (2011)", "Chile (2013)", "Japan (2014)",
               "United States (2012)", "Slovenia (2014)", "Canada (2011)",
               "Poland (2010)", "Estonia (2015)", "Luxembourg (2013)",
               "Portugal (2011)"]

    votingPop = [40, 45.7, 52, 53.6, 54.1, 54.2, 54.5, 54.7, 55.1, 56.6]
    regVoters = [49.1, 42, 52.7, 84.3, 51.7, 61.1, 55.3, 64.2, 91.1, 58.9]

    # notice use of `attr` function to make nested attributes
    trace1 = scatter(;x=votingPop, y=country, mode="markers",
                     name="Percent of estimated voting age population",
                     marker=attr(color="rgba(156, 165, 196, 0.95)",
                                 line_color="rgba(156, 165, 196, 1.0)"
```

```
function area1()
    trace1 = scatter(;x=1:4, y=[0, 2, 3, 5], fill="tozeroy")
    trace2 = scatter(;x=1:4, y=[3, 5, 1, 7], fill="tonexty")
    plot([trace1, trace2])
end
area1()
```

```
function dumbell()
    # reference: https://plot.ly/r/dumbbell-plots/
    @eval using DataFrames

    # read Data into dataframe
    nm = tempname()
    url = "https://raw.githubusercontent.com/plotly/datasets/master/school_earnings.
    download(url, nm)
    df = readtable(nm)
    rm(nm)

    # sort dataframe by male earnings
    df = sort(df, cols=[:Men], rev=false)

    men = scatter(x=df[:School], x=df[:Men], mode="markers", name="Men",
```

```
function subplots1()
    p1 = linescatter1()
    p2 = linescatter3()
    p3 = area1()
    p4 = dumbell()
    p = [p1 p2; p3 p4]
    p.plot.layout["showlegend"] = false
    p.plot.layout["width"] = 1000
    p.plot.layout["height"] = 600
    p
end
subplots1()
```

STATISTICAL PLOTTING

Create a Julia notebook called stats_plotlyjs

IMPORT

```
using PlotlyJS
```

```
function grouped_bar_example()
    trace1 = bar(;x=["giraffes", "orangutans", "monkeys"],
                 y=[20, 14, 23],
                 name="SF Zoo")
    trace2 = bar(;x=["giraffes", "orangutans", "monkeys"],
                 y=[12, 18, 29],
                 name="LA Zoo")
    data = [trace1, trace2]
    layout = Layout(;barmode="group")
    plot(data, layout)
end
grouped_bar_example()
```

```
function stacked_bar_example()
    trace1 = bar(;x=["giraffes", "orangutans", "monkeys"],
                 y=[20, 14, 23],
                 name="SF Zoo")
    trace2 = bar(x=["giraffes", "orangutans", "monkeys"],
                y=[12, 18, 29],
                name="LA Zoo")
    data = [trace1, trace2]
    layout = Layout(;barmode="stack")
    plot(data, layout)
end
stacked_bar_example()
```

```
function two_hists()
    x0 = randn(500)
    x1 = x0+1

    trace1 = histogram(x=x0, opacity=0.75)
    trace2 = histogram(x=x1, opacity=0.75)
    data = [trace1, trace2]
    layout = Layout(barmode="overlay")
    plot(data, layout)
end

two_hists()
```

JOINT

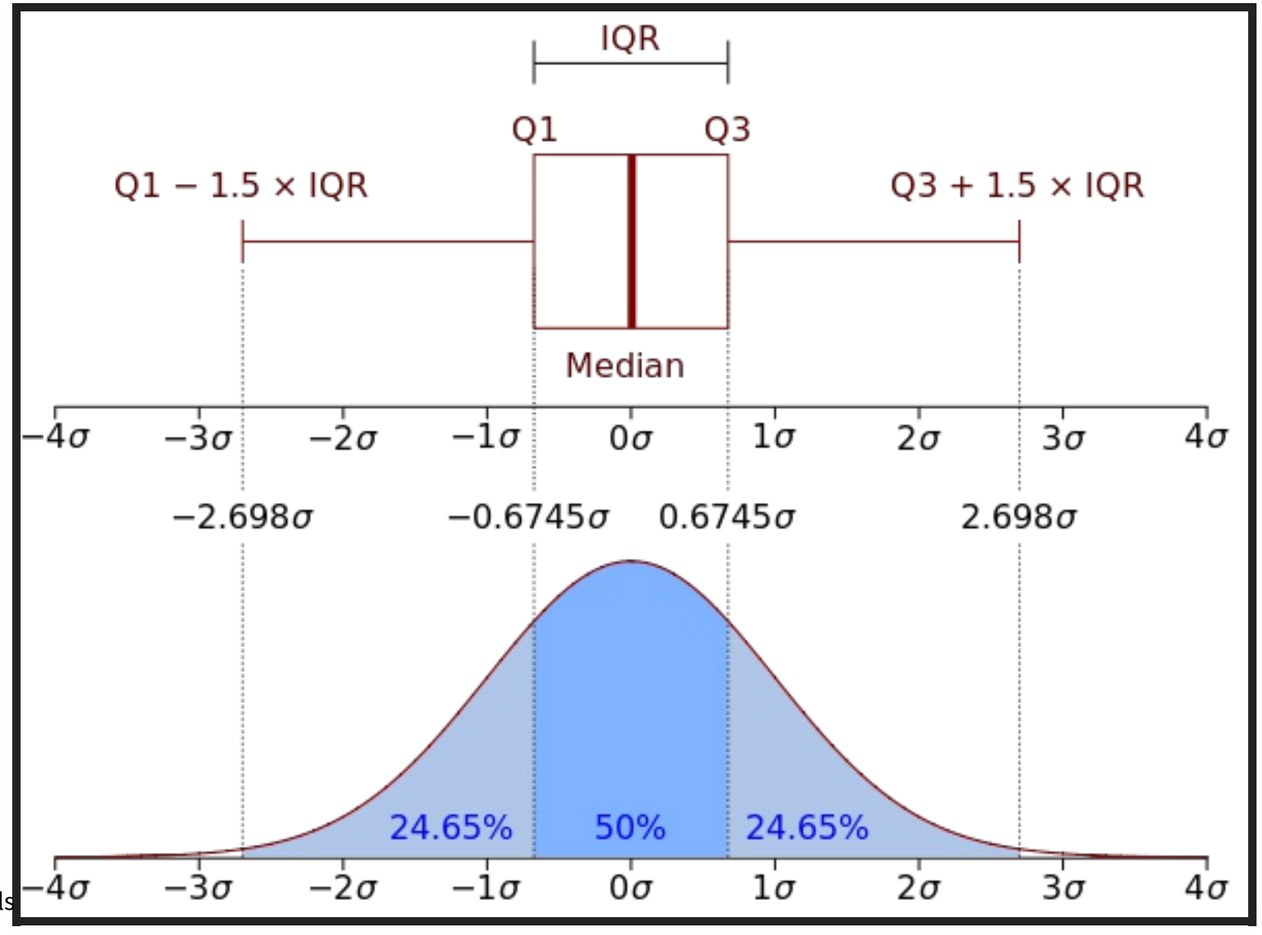
```
function hist_2d_example()
    M = randn(1000,2);
    M[:,2] -= 0.7M[:,1] + 2;

    trace1 = histogram2d(x = M[:,1], y=M[:,2],
                         colorscale = [["0" , "rgb(0,225,100)"],["1", "rgb(100,0,200)"]],
                         xaxis = "x",
                         yaxis = "y2",)


    p1 = plot([trace1])
end
hist_2d_example()
```

BOX (WHISKER) PLOTS

A box plot is a convenient way of graphically depicting numerical data through their quartiles (see figure).



```
function box4()
    x0 = ["day 1", "day 1", "day 1", "day 1", "day 1", "day 1",
          "day 2", "day 2", "day 2", "day 2", "day 2", "day 2"]
    trace1 = box(;y=[0.2, 0.2, 0.6, 1.0, 0.5, 0.4, 0.2, 0.7, 0.9, 0.1, 0.5, 0.3],
                 x=x0,
                 name="kale",
                 marker_color="#3D9970")
    trace2 = box(;y=[0.6, 0.7, 0.3, 0.6, 0.0, 0.5, 0.7, 0.9, 0.5, 0.8, 0.7, 0.2],
                 x=x0,
                 name="radishes",
                 marker_color="#FF4136")
    trace3 = box(;y=[0.1, 0.3, 0.1, 0.9, 0.6, 0.6, 0.9, 1.0, 0.3, 0.6, 0.8, 0.5],
                 x=x0,
                 name="carrots",
                 marker_color="#FF851B")
```

```
function box9()
    xData = ["Carmelo<br>Anthony", "Dwyane<br>Wade", "Deron<br>Williams",
            "Brook<br>Lopez", "Damian<br>Lillard", "David<br>West",
            "Blake<br>Griffin", "David<br>Lee", "Demar<br>Derozan"]

    _getrandom(num, mul) = mul .* rand(num)

    yData = Array[
            _getrandom(30, 10),
            _getrandom(30, 20),
            _getrandom(30, 25),
            _getrandom(30, 40),
            _getrandom(30, 45),
            _getrandom(30, 30),
            _getrandom(30, 20),
```

```
function clusters()
    @eval using Distributions
    x0 = rand(Normal(2, 0.45), 300)
    y0 = rand(Normal(2, 0.45), 300)
    x1 = rand(Normal(6, 0.4), 200)
    y1 = rand(Normal(6, 0.4), 200)
    x2 = rand(Normal(4, 0.3), 200)
    y2 = rand(Normal(4, 0.3), 200)

    data = [scatter(;x=x0, y=y0, mode="markers"),
            scatter(;x=x1, y=y1, mode="markers"),
            scatter(;x=x2, y=y2, mode="markers"),
            scatter(;x=x1, y=y0, mode="markers")]

    args = [(x0, y0, "blue"), (x1, y1, "orange"), (x2, y2, "green"),
```

DON'T FORGET OTHER LIBRARIES

Plots.jl - is great for complex layouts. Vega.jl - great looking plots but it is in transition at the moment
Gadfly.gl - some plots are easy out of the box

https://github.com/bcbi/julia_tutorials/blob/master/plotting/s

HOME OF THESE NOTES AND NOTEBOOKS

https://github.com/bcbi/julia_tutorials/tree/master/biomedqu
https://github.com/bcbi/julia_tutorials/tree/master/plotting