Tutorial III.III - DataFrames in Julia

Applied Optimization with Julia

Introduction

Welcome to this tutorial on plotting in Julia! We'll be using the powerful Plots.jl package to create beautiful and informative visualizations. Don't worry if you're new to plotting – we'll start with the basics and gradually build up to more advanced techniques.

In this tutorial, you'll learn how to: 1. Create simple plots like line graphs and scatter plots 2. Customize your plots with colors, labels, and styles 3. Add multiple data series to a single plot 4. Save your plots as image files for use in reports or presentations

Follow the instructions, write your code in the designated code blocks, and validate your results with `@assert` statements.

Before we begin, let's make sure you have the necessary packages installed. If you've been following the course, you'll need to install the Plots and StatsPlots packages:

```
import Pkg; Pkg.add(["Plots","StatsPlots"])
```

Now, let's load these packages:

```
using Plots, StatsPlots
```

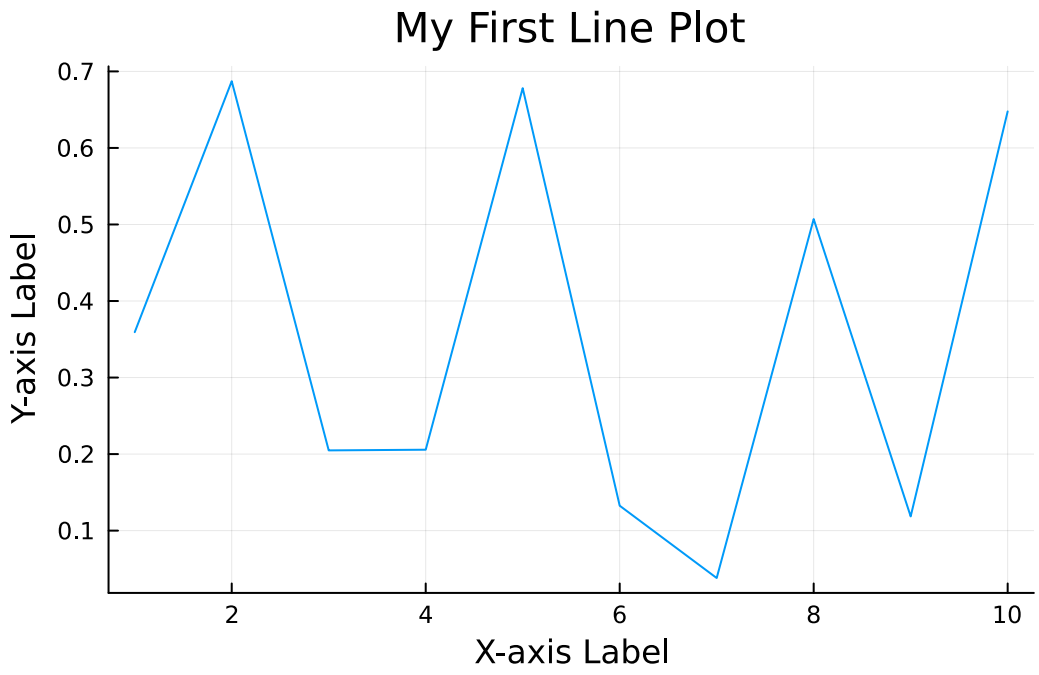
Section 1 - Creating Basic Plots

The Plots.jl package simplifies the process of creating a wide array of plots, from simple line plots to complex 3D visualizations. Let's start with the simplest type of plot: a line plot. We'll create some example data and then plot it.

```
# Create some example data
x = 1:10      # This creates a range of numbers from 1 to 10
y = rand(10)  # This creates 10 random numbers between 0 and 1

# Create a basic line plot
line_plot = plot(
    x, y,
    title="My First Line Plot",
    xlabel="X-axis Label",
    ylabel="Y-axis Label",
    legend=false
)
```

```
# Display the plot
display(line_plot)
```



Let's break down what each part of this code does:

- `plot(x, y, ...)` creates the plot using our x and y data
- `title=...` sets the title of the plot
- `xlabel=...` and `ylabel=...` label the x and y axes
- `legend=false` turns off the legend (we'll use this later)

Exercise 1.1 - Create a Scatter Plot

Now it's your turn! Create a scatter plot using the `scatter()` function instead of `plot()`. Use a range from 1 to 20 for x, and generate 20 random numbers for y.

```
# YOUR CODE BELOW
# Hint: Use x = 1:20 and y = rand(20)
```

```
# Test your answer
@assert @isdefined scatter_plot
println("Great job! You've created your first scatter plot.")
```

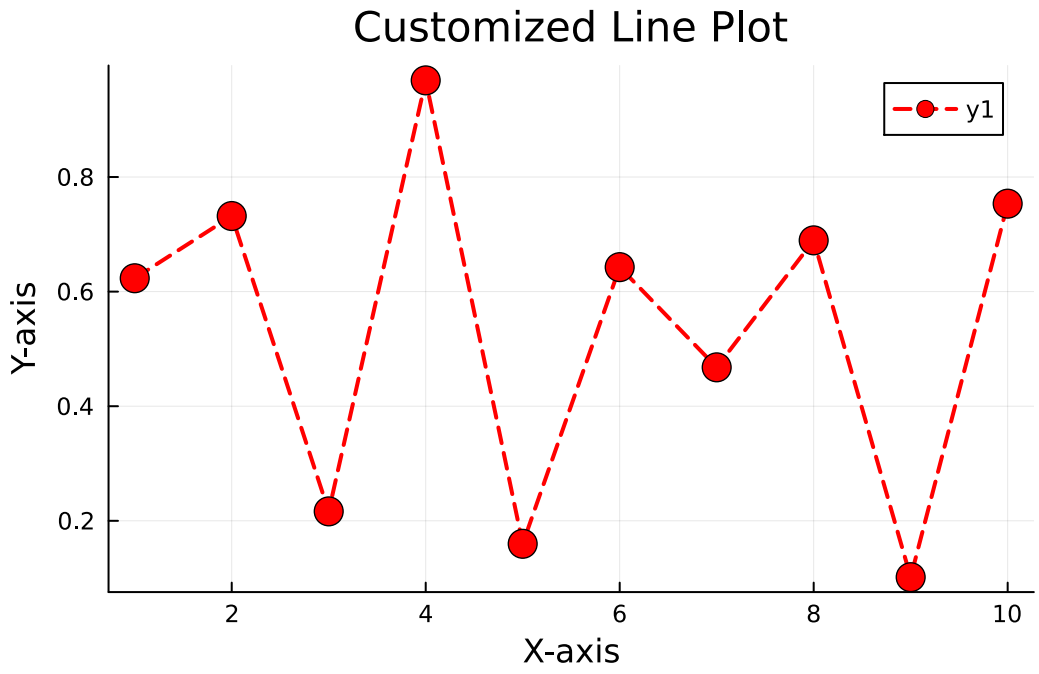
Section 2 - Customizing Plots

One of the best things about Plots.jl is how easy it is to customize your plots. Let's explore some options:

```julia
x = 1:10
y = rand(10)

custom_plot = plot(
    x, y,
    title="Customized Line Plot",
    xlabel="X-axis",
    ylabel="Y-axis",
    line=(:dash, 2),  # Dashed line with width 2
    color=:red,       # Red color
    marker=(:circle, 8)  # Circle markers of size 8
)

display(custom_plot)
```



Exercise 2.1 - Customize a Line Plot

Now it's your turn to get creative! Customize a line plot with your choice of colors, line styles, and markers. Save your masterpiece in the variable `custom_line_plot`.

```julia
# YOUR CODE BELOW
# Hint: Try different line styles (:dash, :dot), colors (:blue, :green),
and markers (:star, :diamond)
```

```julia
# Test your answer
@assert @isdefined custom_line_plot
println("Excellent! You've created a custom line plot.")
```

> **Note**
>
> Feel free to experiment with different options. There's no "right" answer here – it's all about what looks good to you!

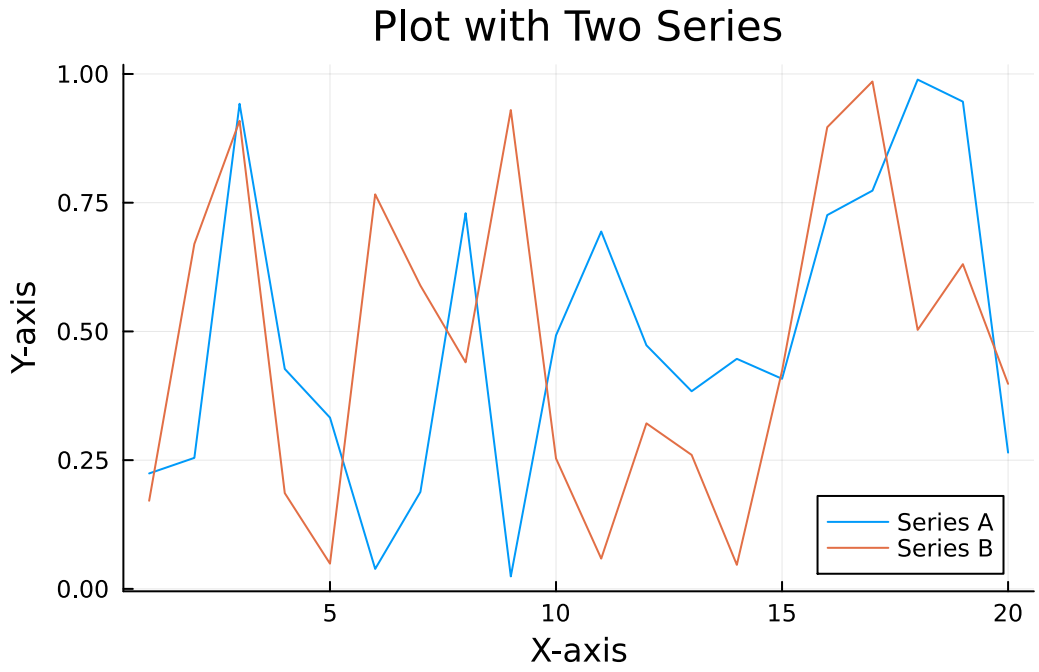
Section 3 - Adding Multiple Series to a Plot

Now, let's learn how to add multiple data series to a single plot:

```
x = 1:20
a = rand(20)
b = rand(20)

multi_plot = plot(
    x, a,
    title="Plot with Two Series",
    xlabel="X-axis",
    ylabel="Y-axis",
    label="Series A"
)
plot!(multi_plot,
    x, b,
    label="Series B"
)

display(multi_plot)
```



> **Note**
>
> The `plot!()` function (with an exclamation mark) adds to an existing plot instead of creating a new one. In addition, we used the label here to add a legend to the plot.

Exercise 3.1 - Create a Multiple Series Plot

NYour turn! Create a plot called `multi_series_plot` with three data series `y1`, `y2`, and `y3`. Make sure to give each series a different color and label.

```
# YOUR CODE BELOW
# Hint: Use plot() for the first series, then plot!() for the second and
third
```

```
# Test your answer
@assert @isdefined y1
@assert @isdefined y2
@assert @isdefined y3
@assert @isdefined multi_series_plot
println("Fantastic! You've created a plot with multiple series.")
```

Section 3 - Saving Plots to Files

Plots.jl supports saving your plots to various file formats including PNG, SVG, and PDF, enabling you to use your plots outside of Julia. The function to save plots is `savefig()`, the first argument is the plot itself and the second argument is the `path/filename` format as string. Replace `path` with the path, the `filename` with the actual name and `format` with the file format, e.g. `pdf`, `png`, … . For example, if you want to save your file as PDF, you would just name it `path/filename.pdf`. For example:

```
savefig(plot_name, "path/filename.png")
```

This saves the plot as a PNG file.

Exercise 3.1 - Save a Plot to a File

Save your `multi_series_plot` as a PNG file named “saved_plot.png” in the “ExampleData” folder.

```
# YOUR CODE BELOW
# Don't forget to use the @__DIR__ macro to get the correct file path!
```

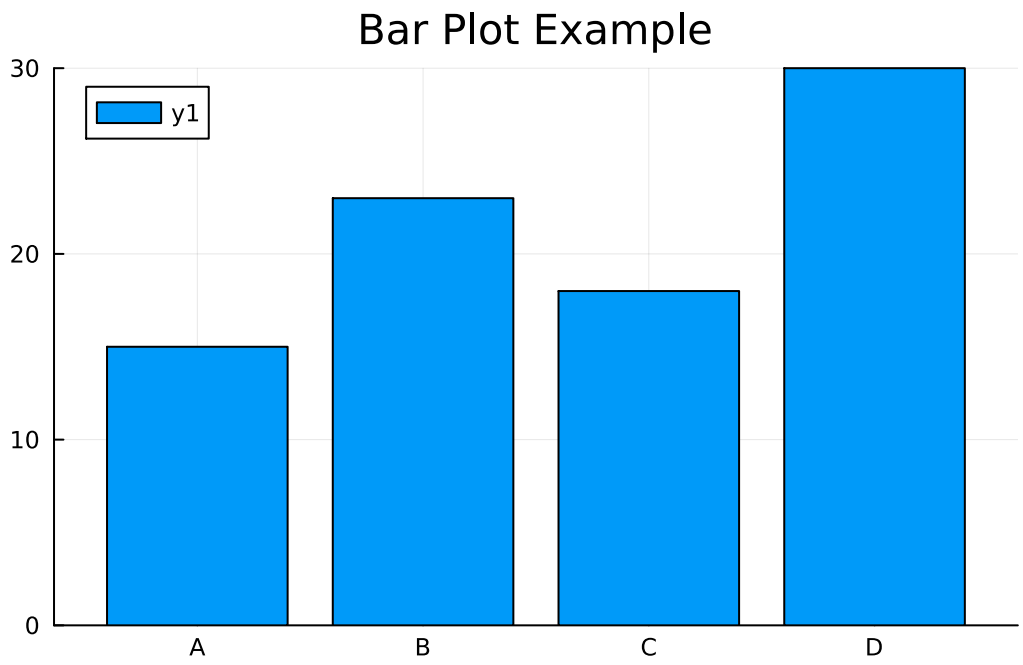
```
# Test your answer
@assert isfile("$(@__DIR__)/ExampleData/saved_plot.png") "File does not
exist yet."
println("Well done! You've saved your plot as an image file.")
```

Section 4 - Advanced Plotting Techniques

Let's explore some other common plot types. While you don't have to solve any task here, the code might come in handy later on during the course as recipe.

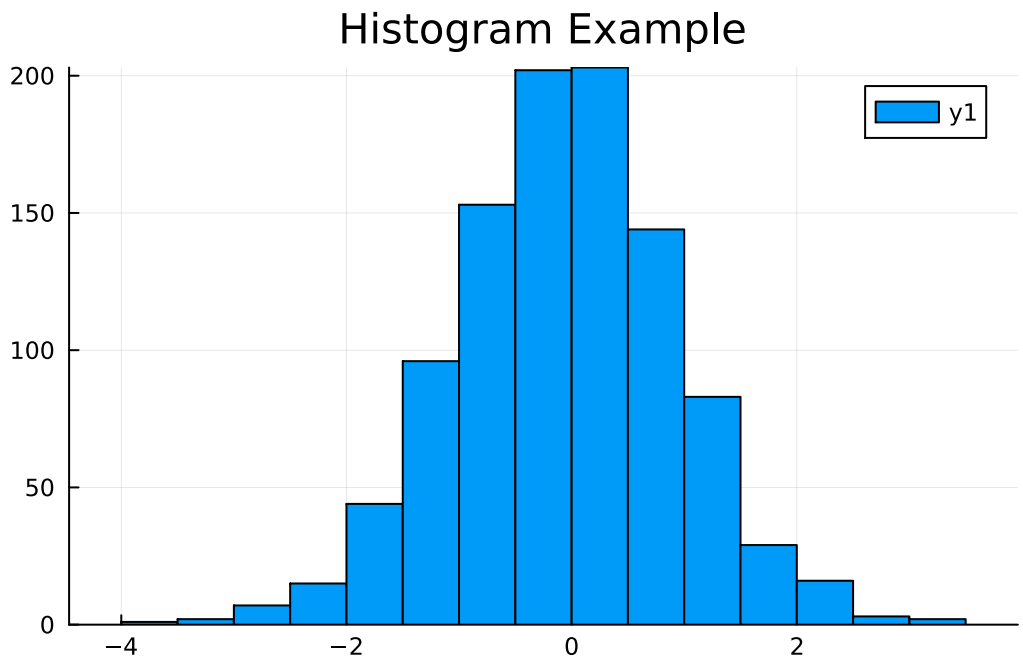
Bar Plot

```
# Bar plot
x_categories = ["A", "B", "C", "D"]
y_values = [15, 23, 18, 30]
bar_plot = bar(
    x_categories,
    y_values,
    title="Bar Plot Example"
)
display(bar_plot)
```



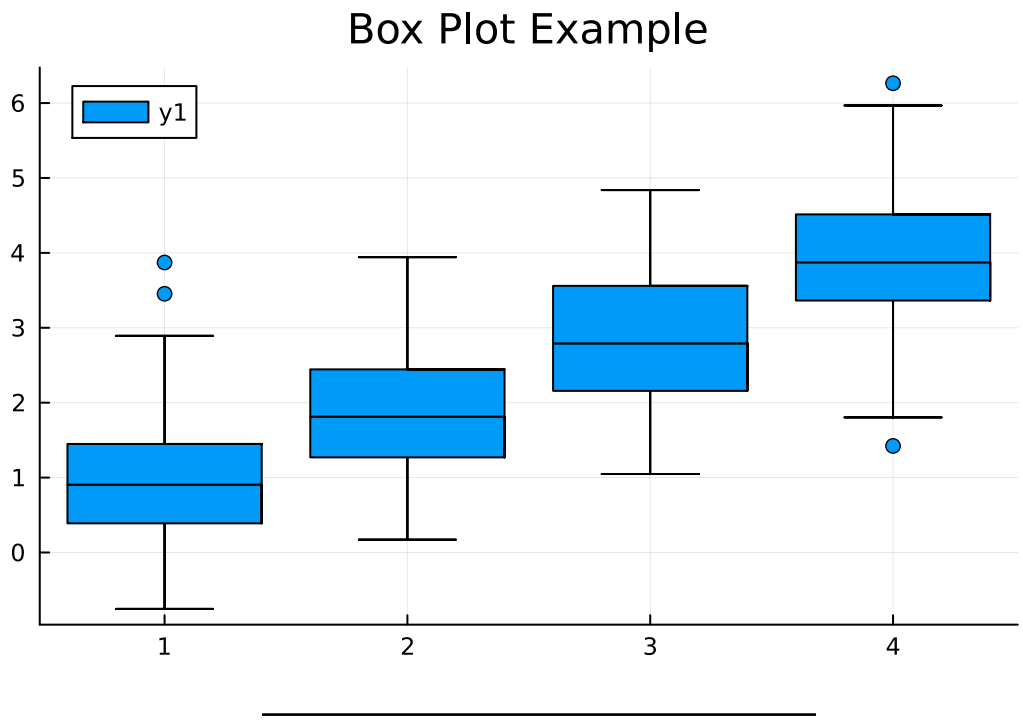
Histogram

```
data = randn(1000)
hist_plot = histogram(
    data,
    bins=30,
    title="Histogram Example"
)
display(hist_plot)
```



Box Plot

```
# Box plot
group = repeat(1:4, inner=50)
y = randn(200) .+ group
box_plot = boxplot(
    group,
    y,
    title="Box Plot Example"
)
display(box_plot)
```



Conclusion

Fantastic! You've completed the tutorial on basic plotting in Julia. You've learned how to create basic plots and customize and save them. Continue to the next file to learn more.

Solutions

You will likely find solutions to most exercises online. However, I strongly encourage you to work on these exercises independently without searching explicitly for the exact answers to the exercises. Understanding someone else's solution is very different from developing your own. Use the lecture notes and try to solve the exercises on your own. This approach will significantly enhance your learning and problem-solving skills.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions.

Later, you will find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them in next week's tutorial. To access the solutions, click on the Github button on the lower right and search for the folder with today's lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. But please remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.